



USING COVERT MEANS
TO ESTABLISH CYBERCRAFT
COMMAND AND CONTROL

THESIS

Bradley D. Sevy, Captain, USAF

AFIT/GCS/ENG/09-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

USING COVERT MEANS
TO ESTABLISH CYBERCRAFT
COMMAND AND CONTROL

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science


Bradley D. Sevy, B.S.C.S.
Captain, USAF

March 2009

USING COVERT MEANS
TO ESTABLISH CYBERCRAFT
COMMAND AND CONTROL

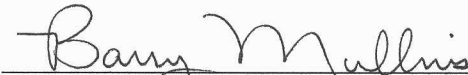
Bradley D. Sevy, B.S.C.S.
Captain, USAF

Approved:



Lt Col J. Todd McDonald, Ph.D.
(Chairman)

12 MAR 09
date



Dr. Barry Mullins (Member)

12 Mar 09
date



Dr. Gilbert Peterson (Member))

12 MAR 09
date

Abstract

With the increase in speed and availability of computers, our nation's computer and information systems are being attacked with increased sophistication. The Air Force Research Laboratory (AFRL) Information Directorate (RI) is researching a next generation network defense architecture, called Cybercraft, that provides automated and trusted cyber defense capabilities for AF network assets. This research we consider the issues to protect or obfuscate command and control aspects of Cybercraft. In particular, we present a methodology to hide aspects of Cybercraft platform initialization in context to formation of hierarchical, peer-to-peer groups that collectively form the Cybercraft network. Because malicious code networks (known as botnets) currently manifest many properties of obfuscating command and control sequencing, we evaluate and consider our proposed methodology in light of leading bot detection algorithms. This research subjects Bothunter to a series of tests to validate these claims. We use a leading bot detection utility, Bothunter, and an ARP validation tool, XArp, to build a case for the effectiveness of our approach. We present three scenarios that correlate to how we believe Cybercraft platforms integrate in the future and consider stealthiness in terms of these representative tools. Our research gives emphasis on measurable hiding related to the Cybercraft initialization sequence, and we show how common network protocols such as ARP, HTTP, and DNS may be modified to carry C2 commands while evading common detection methods found in current tools.

Acknowledgements

First and foremost, I owe a large debt of gratitude to my loving wife. I couldn't have made it through without your love and support. Next I would like to thank Lt Col McDonald for his guidance and direction, and for helping me through the more frustrating times. Third I would like to thank my children for their understanding that daddy had to be left alone "for just a little while longer". Finally I would like to thank my friends and family who bolstered me with their prayers and support. Thank You!

Bradley D. Sevy

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
Table of Contents	vi
List of Figures	viii
List of Tables	x
List of Abbreviations	xi
 I. Introduction	 1
1.1 Research Motivation	1
1.2 Research Statement	2
1.2.1 Research Contribution	2
1.3 Thesis Organization	3
 II. Related Work	 4
2.1 Cybercraft	5
2.1.1 Cybercraft Operation Model	7
2.1.2 Trust	10
2.1.3 Trust Trap	13
2.2 Malicious Software	14
2.2.1 Viruses	15
2.2.2 Worms	15
2.2.3 Rootkits and back doors	16
2.2.4 Bots and Zombies	16
2.2.5 Trojan horses	17
2.3 Malware Taxonomy	17
2.3.1 Type 0 Malware	19
2.3.2 Type I Malware	19
2.3.3 Type II Malware	20
2.3.4 Type III Malware	21
2.4 Virtualization	22
2.4.1 Blue Pill	24
2.5 Cybercraft as a Bot	25
2.5.1 Bot Command and Control	26
2.6 Cybercraft as a Peer-to-Peer (P2P) Network	28

	Page
2.6.1 Chord	29
2.6.2 Encryption	30
2.7 Cybercraft in the Face of the Enemy	30
2.7.1 Byzantine Generals	31
2.8 Bot Hunter	34
2.9 Needle in a Haystack	35
III. Experimentation	38
3.1 Problem Definition	38
3.2 Hypothesis and Goals	38
3.3 Setup	40
3.3.1 Measuring Stealth	41
3.4 Approach	42
3.4.1 Assumptions	44
3.5 Scenarios	48
3.5.1 Scenario I	49
3.5.2 Scenario II	50
3.5.3 Scenario III	51
3.6 Summary	51
IV. Results and Analysis	52
4.1 Testing Bothhunter	52
4.1.1 Bothhunter Experiments	53
4.2 Testing Initialization Sequence	60
4.3 Significance	62
4.4 Summary	63
V. Conclusions and Recommendations	64
5.1 ARP Validation	64
5.2 Future Work	66
5.3 Conclusion	67
Bibliography	68

List of Figures

Figure		Page
2.1	Artist's rendition of a Cybercraft	5
2.2	Cybercraft Operational Concept [16]	7
2.3	Boyd's OODA Loop Sketch [3]	9
2.4	Ants taking poison back with them to their nests [42]	12
2.5	Common trust structure that opens vulnerabilities of insider sabotage or <i>TrustTrap</i> [30]	13
2.6	Type 0 malware. Does not subvert OS directly [35]	18
2.7	Type 1 malware. Subverts OS code [35]	20
2.8	Type 2 malware. Subverts OS data [35]	21
2.9	Severed and resworn DKOM'd link list hides rootkit process [29]	22
2.10	Type 3 malware. Is completely separate from OS [35]	23
2.11	SubVirt before and after infestation [20]	24
2.12	Blue Pill before and after infestation	24
2.13	Two types of command and control methods [13]	26
2.14	The Chord system showing a mapping of items to nodes [41] .	28
2.15	Two honest Generals passing verbal messages with one traitor General	32
2.16	Two honest Generals passing signed written messages with one traitor General	33
2.17	BotHunter Architecture [12]	34
3.1	Proposed network for experiment	41
3.2	Screen shot from Bothunter [17]	43
3.3	Showing that a Virtual Machine Based Rootkit is closer to the hardware than the Guest Operating System	46
3.4	CyberCraft displayed as a Chord Ring of Virtual Machine Monitors communicating with each other	48

Figure		Page
3.5	First scenario, only connection is using HTTP	49
3.6	Second scenario, Cybercraft gains plugs into local Chord Structure using modified ARP messages	50
3.7	Thrid scenario, Cybercraft platform connection to outside C2 structure via DNS messages	51
4.1	Screenshot of a file that was downloaded and executed using Sdbot	56
4.2	Graph showing the drop in profiles as the packet number increased	57
4.3	Screenshot showing webpage that could have C2 information .	61
5.1	Screenshot of XArp a ARP validation utility from [25]	65

List of Tables

Table		Page
3.1	List of Bots Chosen for Experiment	42
3.2	Table of Bothuner tests	44
3.3	Table showing first test criteria	45
3.4	Table showing second test criteria	45
4.1	Table with bot detection results	54
4.2	Table showing C2 results	59
4.3	Table showing Cybercraft initialization sequence results	60

List of Abbreviations

Abbreviation		Page
C2	Command and Control	2
UAV	Unmanned Aerial Vehicle	7
IO	Information Operations	7
DDoS	Distributed Denial of Service Attack	8
ROEs	Rules of Engagement	8
OODA	Observe Orient Decide Act	9
OS	Operating System	18
API	Application Programming Interface	19
DLL	Dynamic-Link Library	19
DKOM	Direct Kernel Object Manipulation	20
VM	Virtual Machine	21
VMBR	Virtual Machine Based Rootkit	21
VMM	Virtual Machine Monitor	21
HAV	Hardware-Assisted Virtualization	24
IRC	Internet Relay Chat	27
IM	Instant Message	27
HTTP	Hypertext Transfer Protocol	27
IDS	Intrusion Detection System	27
HP2P	Hierarchical Peer-to-Peer	28
DHT	Distributed Hash Table	29
SMC	Secure Multi-Party Computations	30

USING COVERT MEANS TO ESTABLISH CYBERCRAFT COMMAND AND CONTROL

I. Introduction

In his commentary on Sun Tzu's The Art of War, Zhuge Liang, one of China's greatest military strategists and philosophers said [6]:

“...high walls and deep moats do not guarantee security, while strong armor and effective weapons do not guarantee strength. If opponents want to hold firm, attack where they are unprepared; if opponents want to establish a battlefield, appear where they do not expect you.”

Zhugue Liang's observations of warfare ring as true today as when he penned them over 1800 years ago. Although there have been many advancements in military technology, no amount of ingenuity has changed the core property of war that a successful attack is one that goes against an enemies weakness, and a strong defense is contingent on hiding those weakness from the enemy. However, much has changed with regard to where the battles take place and the speed at which they are fought. Today, key military systems are constantly bombarded on the cyber battlefield. Hundreds of attacks are waged against strategic computer and information systems with varying sophistication from trivial to elite.

1.1 Research Motivation

To better prepare our nation for cyber war, the Air Force Research Lab's Information Directorate has launched a multi-year research project to bolster cyber defenses.

Named the Cybercraft project, this effort seeks to establish a Cybercraft that automates tasks which increases the defenses of military computer and information networks [16].

On 15 September 2008 the Air Force’s top leaders released a joint letter updating the Air Force Mission. The new mission of the Air Force is: [9]

”The mission of the United States Air Force is to fly, fight and win ... in air, space and cyberspace.”

Cyberspace again has been stressed by Air Force top leadership as vital to the Air Force mission. This work contributes to the Air Force mission to ‘win’ the fight in cyberspace by furthering the work that has already been made on the Air Force’s Cybercraft Project. Specifically this research will develop a methodology for a stealthy initialization sequence for when a new Cybercraft comes online and plugs into a command and control channel.

1.2 Research Statement

This research presents a method for stealthy Cybercraft initialization sequence required to gain command and control (C2).

1.2.1 Research Contribution. It is predicted that 10% of all computers on the Internet are being controlled by a botnet. [28] This creates a large concern for the Air Force and its mission to protect its computer systems. Additionally the Air Force needs to protect the methods, techniques and tools used in defense of these malicious threats. Much work has already been done in the field of botnet detection and prevention. The first contribution is to cast the Cybercraft C2 problem as a botnet detection problem. Furthermore, we propose concrete scenarios based on current botnet detection systems to evaluate possible techniques of stealthy initialization. Finally, we provide an evaluation of Bothunter and other tools based on our analysis.

1.3 Thesis Organization

This document is divided into five chapters. Chapter II gives the background that is needed to understand our research and methodology. Chapter III outlines my proposed methodology, our hypotheses and our approach to setting up an experiment to test those hypothesis. Chapter IV gives a detailed evaluation of our methodology experiment. Chapter V gives recommendations for future work and concludes.

II. Related Work

With the frequency and the level of sophistication of computer attacks on the rise there is a current need to revamp the defense of computer and information systems. The Department of Defense repels hundreds of computer attacks every day, and on a global scale it is estimated nearly 10% of all nodes connected to the Internet belong (unknowingly) to malicious multi-agent systems (bots). [28] The fact that 10% of the worlds computers could be controlled by someone with malicious intent is a strong enough threat against our nations security and well-being that a project has been developed to mitigate that threat.

This chapter summarizes the background information necessary to better understand our research regarding the development of a stealthy Cybercraft initialization sequence. It is divided into nine sections. Section 1 *Cybercraft* more fully explains the Cybercraft project, including its operational model. Because this research is based around the Cybercraft project, it is important to understand the models that serve as the foundation of the research. Section 2 *Malicious Software* briefly outlines many forms of malware currently in practice. The characteristics and lessons learned from these malicious software groups will be used to achieve a stealthy Cybercraft initialization. Section 3 *Malware Taxonomy* describes a taxonomy to describe the weaknesses specific malware types use to exploit systems. This is important to distinguish because the type of vulnerability used has consequences on the level of effort required to reveal the exploit. Section 4 introduces a rootkit that uses a virtual machine monitor, highlighting a significant aspect of our methodology for stealthy Cybercraft initialization. Section 5 *Cybercraft as a Bot* depicts command and control characteristics of a botnet. It also compares a botnet command and control structure to that of Cybercraft. Section 6 *Cybercraft as a P2P Network* describes the characteristics of a Cybercraft initializing and joining a peer network. This is used in our methodology as a way to successfully and stealthily gain command and control. Section 7 *Cybercraft in the Face of the Enemy* assumes that Cybercraft could fall victim to malicious soft-



Figure 2.1: Artist's rendition of a Cybercraft

ware. It relates the problem of enemies interfering with Cybercraft to the Byzantine Generals problem classically used to describe overcoming fault tolerance issues in a distributed system. Section 8 *Bothunter* introduces the Bothunter bot detector used to test our stealthy initialization sequence and command and control methodology. Finally, Section 9 *Needle in a Haystack* describes the challenge of sorting through legitimate traffic patterns to find malicious information. The methodology for stealthy Cybercraft initialization is based on modifying common information and hiding in plain sight just like a needle in a haystack.

2.1 *Cybercraft*

The basis of our research is to show that a Cybercraft can stealthily gain command and control even in the presence of modern day bot detectors. This section describes in greater detail the characteristics of a Cybercraft.

The Cybercraft project is an undertaking headed by the Air Force Research Lab, and its mission is “to automate cyber defense responses based on fused trust-based situational awareness data and operator-defined rules of engagement” [16]. The Cybercraft project is still in its infancy and is continuously growing its list of requirements and specifications.

The Cybercraft’s basic premise is that it is to be a lightweight distributed multi-agent system composed of both software and hardware. [19] The Cybercraft’s mission will be to defend and sustain military networks and assets and as such must [19,28]:

- monitor systems and respond in near real time
- respond according to current policies
- provide feedback to human operators of mission status
- support varying levels of autonomy
- be dynamically configurable
- follow a command structure

Because the Cybercraft is to be used by the military, it is expected to be scalable, redundant and have a well defined command and control component. Undertaking such a project brings with it many challenges. Dr. Phister [32] raised several important questions that help define the role Cybercraft needs to play in defense of our information systems. Some of the questions he asked were: What would the mission’s of the Cybercraft be? How do you control the Cybercraft? How can we trust the Cybercraft to do the right thing? These are all important questions, especially for the operators and commanders that will be relying on the Cybercrafts. The answers to these question are beyond the scope of this paper, however, this research does intend to further the answer to how a Cybercraft should initialize, how it can do so stealthily and how it can gain command and control.



Figure 2.2: Cybercraft Operational Concept [16]

2.1.1 Cybercraft Operation Model. One of the concepts behind Cybercraft is that a Cybercraft can be resourced to carry out diverse missions. This is achieved by executing different dynamic code or payloads. This can be viewed similarly to current unmanned aerial vehicles (UAV) flown today. Current UAVs in operations today can be fitted with different sensors and/or weapons payloads to execute a variety of missions. However, no matter the payload the airframe of the UAV doesn't change from mission to mission. Similarly to the UAV airframe, the Cybercraft *platform* does not change from one mission to another. Instead the Cybercraft can be fitted with different dynamic payloads. This gives Cybercraft flexibility in the missions it can perform while maintaining a stable base of static code. For the purposes of our research we are interested in the Cybercraft platform; and any further reference to Cybercraft should be taken to refer to the Cybercraft platform.

One of the US Air Force's seventeen Air and Space Power functions is Information Operations (IO). According to USAF doctrine, information operations "are actions taken to influence, affect, or defend information, systems, and/or decision-

making to create effects across the battlespace.” [43] There are many scenarios where a Cybercraft could be used to fulfill an Information Operation mission. Take for example the scenario of a distributed denial of service attack (DDoS). Under a DDoS attack many computers from around the globe converge on a specific target in an attempt to inundate the target’s resources and bandwidth so completely that they crash or slow to a stop. A DDoS could happen so fast that by the time it would take a human to recognize a DDoS and to react to it, the damage could already be done. This is when Cybercraft is important. Figure 2.2 displays the operational concept for a Cybercraft mission such as the DDos attack previously outlined. As currently defined a Cybercraft would not be activated without a specific set of Rules of Engagement (ROEs). These rules, authorized by someone of authority, would be set in place prior to any action on the part of the Cybercraft. To illustrate the example, a commander may authorize a Cybercraft to patrol the borders of the network and keep a vigilant eye on the network resources.

Prior to being initialized, we envision one or more Cybercraft be installed on computational nodes. The installation procedure will be accomplished by technicians or operators. Once a Cybercraft connects, either at the software or hardware level, it must then join the aggregate hierarchy, known as the Cybercraft ”cloud”. This communication is necessary for a Cybercraft to plug into proper command and control channels. This initialization sequence could generate a high level of computer communication traffic, and if not properly executed could attract much attention. Since one of the goals of the Cybercraft system is self-defense, a key facet of a defensive system is to be stealthy, or in other words, to remain undetected [14].

This research aims to quiet this communication and make the initialization as stealthy as possible. Just like prisoners, it attracts less attention to whisper to a neighbor than to pass communications in and out of prison. In this way if a Cybercraft platform is connected to a command and control channel already, future Cybercraft

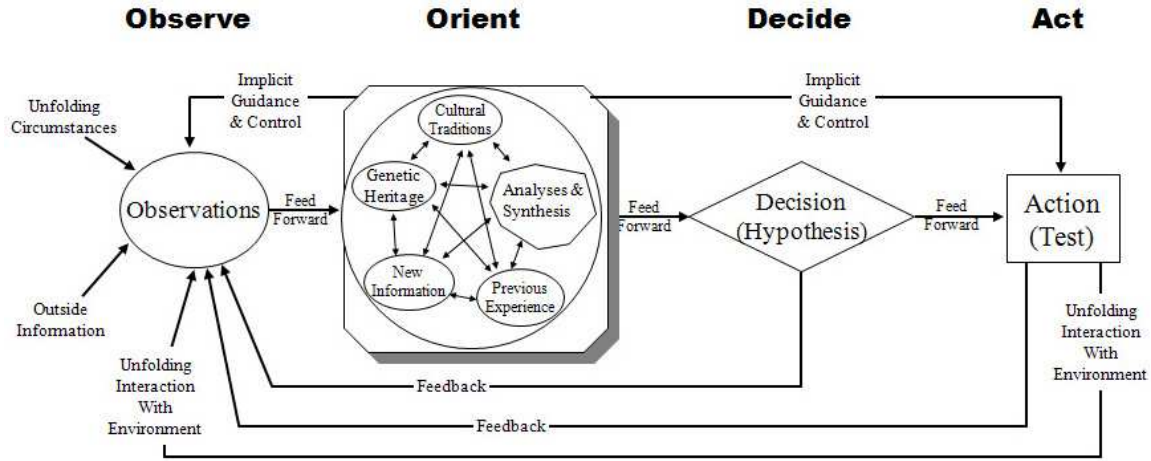


Figure 2.3: Boyds OODA Loop Sketch [3]

platforms can leverage this and limit the amount of noise or attention that unnecessary communications could warrant.

Once initialized and operational, a Cybercraft would be a functioning member of a command and control structure. As a member of the command and control structure the Cybercraft would receive payloads and policies to execute from other sources. These communications would be encrypted covert channels (at a minimum). Our interest is the initial communication sequence which all Cybercraft platforms must use to join the command and control channel or “cloud”.

This Cybercraft Operational Concept parallels the US Air Force OODA Loop concept that has proven so successful in air-to-air combat missions (see Figure 2.3). The OODA or Observe - Orient - Decide - Act loop was first developed to express to air-to-air fighter pilots that in a dogfight the pilot that was faster at realizing what was going on and reacting to it was the pilot that was going to survive the encounter. With one iteration of the OODA loop, a pilot observes what is going on around him, make a decision, and then carry out that decision. Whatever action the pilot chooses has ramifications and consequences. This requires the pilot to carry out another it-

eration of the OODA loop, the cycle continues until the struggle is over. The pilot that is able to get inside the others OODA loop, or rather have a tighter OODA loop than the opponent is likely to be the victor.

This same concept can be applied to the cyber domain of information and computer security. Since the medieval days of knights crossing moats and storming castle gates, success on the battlefield is determined not only by guarding your weak areas, but in how fast you can react when they are being attacked. In the cyber realm humans start at a severe disadvantage. Modern computing components are capable of making billions of calculations every second. One current computer attacker has developed a weapon called Stormbot. The Stormbot is very sophisticated and may have contaminated over 2 million computers. One organization has reported that Stormbot is currently infesting computers at the rate of 900,000 computers monthly. [8] Assuming 2 million computers performing calculations at 1 billion per second equals a force that is capable of 2.0×10^{15} or 2 quadrillion calculations every second. If such a force was unleashed on any network it surely would not stand unless the network was able to react at similar speeds.

2.1.2 Trust. One of the key questions of Cybercraft is how can a commander trust that the Cybercraft is always going to do the right thing? This research stands on the shoulders of several other scholars who have devoted their entire research on the subject of trust. In their respective theses, Hunt and Stevens [15, 40] worked to develop a framework to define trust models for the Cybercraft platform. At the root of their trust models were the underlying components of current and historical data, intrinsic knowledge of the remote agent's abilities, and recommendations from other agents [40]. To summarize their work they use historical data and peer recommendations to achieve a level of trust with another possible unknown entity. This mimics human trust patterns in many ways [15]. Take for example a human trying to find

a new auto mechanic or a new babysitter. One may search the phone book looking for someone who has been established in the community for awhile, thus basing their decision of trust on historical data. One might also query neighbors or friends in the area and ask for recommendations. The level of trust then stems not only from the recommendation given, but also from the level of trust that exists with the person making the recommendation. If someone that is not known to be trustworthy gives an excellent recommendation, it might not be accepted at the same level, if it was given from someone known to be trustworthy.

In order to achieve a mature trust model, historical data and intrinsic knowledge may help, but verification must play a primary role. In his farewell speech to the nation President Ronald Reagan said regarding our position with the post cold war Russian nation [33]:

“We must keep up our guard...as we make it clear that we will continue to act in a certain way as long as they continue to act in a helpful manner. It’s still trust but verify. It’s still play, but cut the cards. It’s still watch closely. And don’t be afraid to see what you see.”

This notion of trust but verify is a key component of a trust model. It is another way of saying I trust a given entity, but only after I have independently verified it. This model of behavior is commonplace in circumstances where even one failure to identify a malicious entity could result in very serious consequences, either in loss of life or loss of resources. Take for example the guards posted at military installations, or the Transportation Security Administration agents that screen people prior to flying on an airplane. They have a “trust but verify” attitude. They trust a person only after they have checked their credentials and completed all security measures that they deem fit. These security measures take place each and every time one wishes to enter a military installation or fly on a plane, regardless of the number of times one has successfully done so previously. These measures are in place because it is impossible to gauge if someone who has flown on an airplane countless times without incident has malicious intent for the next trip planned. Letting such an individual through



Figure 2.4: Ants taking poison back with them to their nests [42]

the security checkpoint based on such historical data or recommendations from peers could spell doom for all of the passengers on the plane.

Since computer and information networks are vital to the security and well being of the United States, computer security guards should take the same attitude as the physical security counterparts. In addition to the previously mentioned scenarios in the physical world, Cybercraft has to deal with the added threat of zombies and logic bombs. This introduces the more refined notion that Cybercraft trust is akin to fusing possibly “dirty” or malicious data. To illustrate this point imagine a healthy colony of ants (see Figure 2.4). Ants have established protection mechanisms from other intruding insects, and if challenged the ants will swarm on the intruder and attack it. Ants have a requirement to leave the security of their ant hills to accomplish various external tasks, such as foraging for food. A well known method of eradicating ants exploits these scout ants that are out foraging for food. A poison developed to attract ants is placed outside of the ant hill (see Figure 2.4). When the ant comes in contact with the poison, it does not immediately kill it. Instead, the ant is allowed to carry the poison back into the ant hill. The ants, knowing that he is one of their

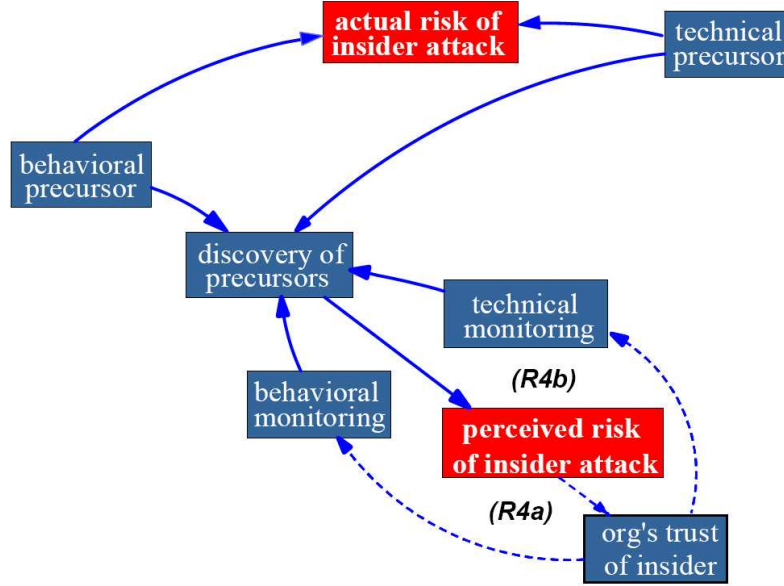


Figure 2.5: Common trust structure that opens vulnerabilities of insider sabotage or *TrustTrap* [30]

scout ants, let it back into the ant hill where he infects numerous more ants in the heart of the ant hill. This attack is successful because the ants have established a trust relationship based on history and peer recommendations. The ant that brought the poison into the ant hill and put the death nail in the community may not have even been aware.

2.1.3 Trust Trap. Another more formal example of trust relationships was conducted by the Software Engineering Institute at Carnegie Mellon [30]. It was the aim of the study to analyze IT sabotage and observe the general characteristics of the individuals that conducted internal sabotage and the circumstances that allowed it to happen. In their work they presented what they call the “*Trust Trap*” (see Figure 2.5). It is pointed out in their work that the actual risk of insider attack is based on the combination of behavioral and technical precursors of the malicious individual. However, many offices and organizations try to foster a trusting relationship between employees, and according to [30]:

“Workplace relationships sometimes shortcut essential behavioral and technical monitoring procedures, or allow them to erode over time due to competing pressures and priorities. Lower levels of monitoring lead to undiscovered precursors, resulting in an overall lower perceived risk”

This trend of trusting the employees and work associates continues, and although the *actual risk* of an individual doing something harmful does not change, the cycle continues and the *perceived threat* continues to diminish until something happens that is so severe that all parties notice.

It is essential then that Cybercraft trust models be created that do not fall victim to the ‘Trust Trap’. A “trust but verify” approach must be taken every time to every Cybercraft. This will increase the difficulty for individuals with malicious intent to sabotage a Cybercraft and send it back into the trusted network, where like the aforementioned ant can use an inside trust to destroy the Cybercraft protections.

2.2 Malicious Software

Malicious software or *malware* is a broad term that is used to define any type of software that is malicious in nature. With the increase of sophistication that malware authors are using to deploy their software, more specific terms are needed to identify different types of malware. This section describes some of the key characteristics of malicious software. In order to create a stealthy methodology for a Cybercraft initialization sequence I have pulled lessons learned from the below categories of malware.

McClure, Scambray and Kurtz [26] break malware down into five sub categories: viruses, worms, rootkits and back doors, bots and zombies and finally trojan horses. Each one of these malware types can be deployed by themselves or in conjunction with one another to serve the purpose of the aggressor. To better understand the

differences in each category a brief explanation will follow.

2.2.1 Viruses. Viruses are the term that most people are familiar with to describe general malware. It is also the name that many of the security vendors have adopted to defend against malware. People purchase anti-virus software to protect against computer viruses, but today's software fights much more than only viruses. In its purest definition a virus is "a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself." [7] In other words viruses are a type of malware that need some other application to serve as the virus' host. It then will attempt to infect other applications with copies of itself. A key characteristic of a computer virus is that it requires interaction [26] to reproduce itself. Viruses can be programmed to attach themselves to a host, then upon some instruction either internally or externally generated carry out almost any code instruction as the host.

2.2.2 Worms. A worm is a piece of software that plants copies of itself in remote, electronically-connected nodes. [21] Worms differ from viruses in that they do not require another application to serve as a host for the worm. Worms also do not require any user intervention to help propagate the worm. In other words worms can self-propagate via a network [26] and cause havoc utilizing system resources rather than modifying the logic of applications. It has been seen that worms are capable of carrying a secondary payload that can execute additional instructions [26]. However many worms simply eat network bandwidth and grow to monopolize network resources. Many worms are programmed to scan their surroundings in order to autonomously propagate. This network vulnerability scanning could draw attention if it was to take place on a network with a vigilant network administrator

2.2.3 Rootkits and back doors. Rootkits are larger than any single piece of software. As the term kit implies, a rootkit is a collection of software application that works together for a common purpose. According to [14] a rootkit is “a set of programs and code that allows a permanent or consistent, undetectable presence on a computer.” This definition really stresses the importance of a rootkit to be stealthy. The name rootkit was derived as many of the first rootkits were developed to provide “root” or full administrator access to a system. Other rootkits have been developed for the purpose of hiding processes, ports or services and also used for software eavesdropping. Some rootkits took the form of packet sniffers, keyloggers or were used to capture email. Because most rootkits are designed to be persistent (survive a computer reboot) and stealthy they are a popular choice for back doors. A back door is a malicious way for someone to gain access to the system without going through proper identification and authentication channels. A rootkit might be written to allow anyone with a particular password access to that computer. This allows the aggressor to gain access to the target system at any time.

2.2.4 Bots and Zombies. Bots started to appear on the computer scene in the early 1990s and were used as a tool to carry out administrative chores over a network on many computers at once [2]. Since then, bots have been developed for malicious purposes and have spawned many new terms. The traditional definition of a bot according to [39] is “bots are simply software programs that perform some action on behalf of a human on large numbers of infected machines.” This network of infected machines are called bot-nets, and the infected machines themselves are called zombies. The individual that has control over the bot-net is referred to as a bot-herder. These bot-herders have been able to gain the resources and control of over a hundred thousand computers at a time [2]. They are then able to use these bot-nets for vary nefarious purposes, some of the most common uses for bot-nets are denial of service floods, vulnerability scanners, e-mail harvesters, email spammers and

allowing the bot herder to surf the Internet without revealing his/her location. Unlike worms and viruses that work autonomously, bots rely on an individual to execute the commands. A more in depth look at the command and control structure of bots can be found in Chapter III §2.5.1

2.2.5 Trojan horses. Software trojan horses, or simply trojans, are software programs “that does something other than, or in addition to, its purported functionality.” [26] Named after the stratagem that allowed the Greeks to finally enter and conquer the city of Troy, software trojans are designed to be appealing, and when unsuspecting users download or install the software the trojan also carries out additional instructions to fulfill a secondary purpose. The secondary purpose could be anything to establishing a backdoor to the system, to disabling anti-virus software to formatting the hard drive. Today trojans have been found in everything from screen-savers, to electronic greeting cards, to fake anti-virus software. Trojans are not only a very effective way to get a user to install malware, but also have a high degree of stealth associated with them. If a user installed a game it would not surprise them to see a game process or registry keys referring to that game. This keeps the user unsuspecting while the trojan is executing all of the malicious code.

2.3 Malware Taxonomy

The previous sections briefly introduced the different types of malware. This section proposes a taxonomy of malware based on the weakness the malware exploits. The taxonomy classifications have strong ramifications on the detectability level of malware. I have considered this taxonomy in our methodology of a stealthy initialization sequence.

Within each type of malware, different approaches could be taken by the malware author to gain access to the system. Each different approach brings with it certain tradeoffs. This taxonomy was formalized by noted security researcher Joanna Rutkowska who uses an alternate definition of malware. Her definition focuses on the specific relationship between the malware and the operating system (OS) and is defined as follows: [29, 35].

“Malware is a piece of code which changes the behavior of either the operating system kernel or some security sensitive applications, without a user consent and in such a way that it is then impossible to detect those changes using a documented feature of the operating system or the application.”

Below four classifications of malware are discussed along with their unique characteristics.

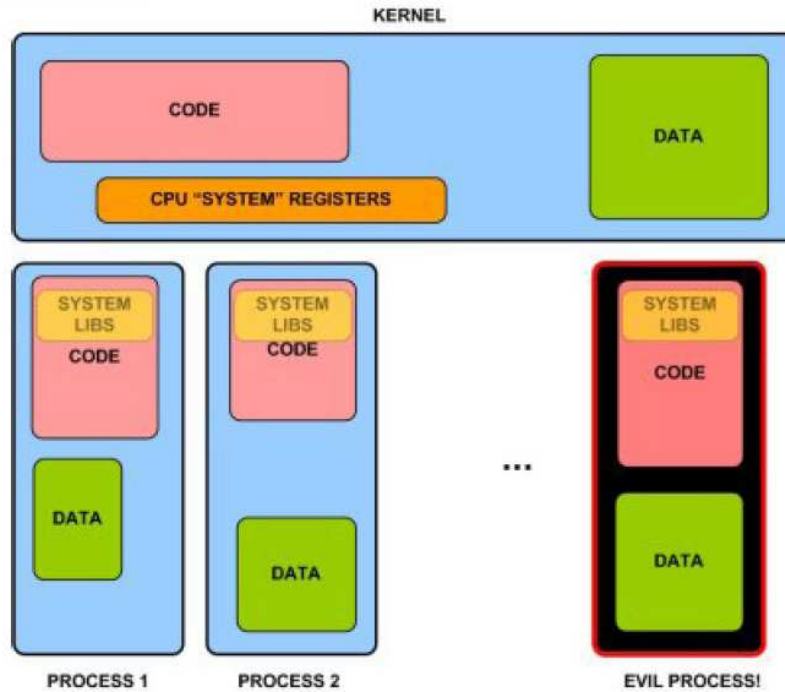


Figure 2.6: Type 0 malware. Does not subvert OS directly [35]

2.3.1 Type 0 Malware. Type 0 malware is defined as malware that does not exploit any weaknesses in the OS, nor does it utilize any undocumented system tricks. Rather it uses documented application programming interfaces (API). This results in code that can be tracked and monitored by the OS. The difference between type 0 malware and all of the other honest programs running on the computer is simply the authors intent (see Figure 2.6). Because type 0 malware does not use tricks to fool the system, it usually will rely on fooling the users. Social engineering is a huge part of type 0 malware. If a malware author can fool a user into downloading and running an application, then there is no need to hide it from the OS. When a user sees the process running in a task list there is no alert, because the user chose to install it. This is typical in many trojan applications. Users think they are getting a new screensaver or Internet browsing toolbar, and are not alarmed to see those same processes running on their systems. However, many programs may lie to the users and be conducting a secondary function such as password stealing or keylogging, but in type 0 malware these secondary functions do not use tricks to go around established OS procedures.

2.3.2 Type I Malware. Type I malware is malware that changes data stored in locations that are not supposed to be changed. This could happen either in memory or in static executable binary files. Traditionally a malware author reverse engineers a binary, or a Dynamic-Link Library (DLL) and introduce changes to the existing executables such that the malware author is able to accomplish a task (see Figure 2.7). The changes could range from disabling security measures or escalating privileges, to jumping and executing new code written by the malware author. Because the exploits for this kind of code are as limitless as the imagination of malicious authors, the best defense against type I malware is simply to ‘know thyself’. This means taking action before a compromise and taking a snapshot of all of the important files on the system. At a later time the user can compare the current files to those in the snapshot and note any differences. Since the files associated with type I malware are not sup-

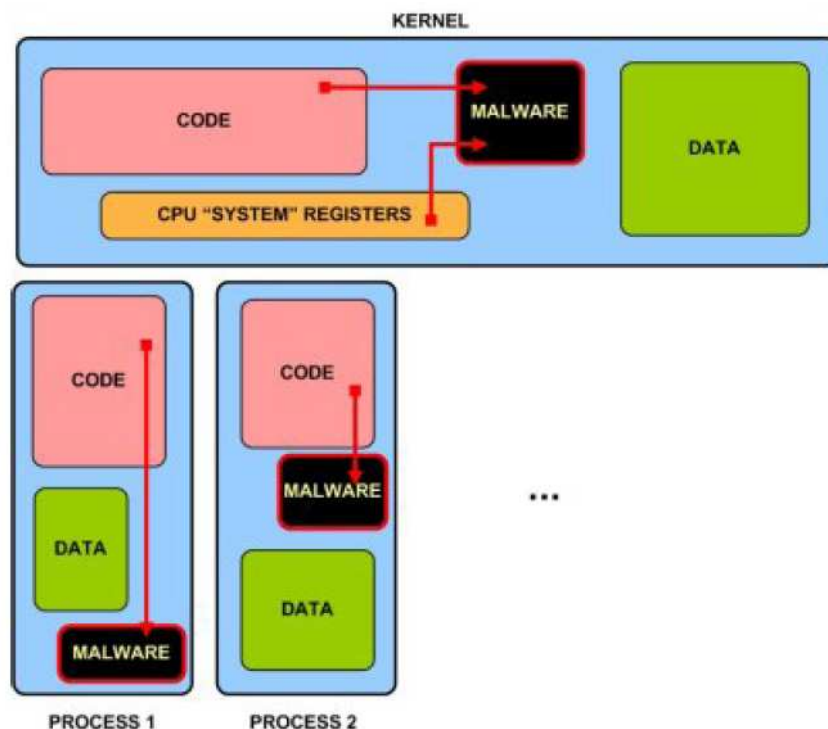


Figure 2.7: Type 1 malware. Subverts OS code [35]

posed to be changed, any differences would stand out as being infected. Several tools exist that work on this premise, including System Virginty Verifier by Rutkowska [34].

2.3.3 Type II Malware. Type II malware is similar to type I malware, except that instead of infecting the code sections of the system that are not supposed to change, type II malware targets the data portions of the system that by necessity are supposed to change often (see Figure 2.8). For example, the data and pointers inside kernel data structures is dynamic and constantly changing. It is possible to change them to hide processes, device drivers, ports, or to even skew forensics. [14] One method popularized in the ‘fu’ rootkit is Direct Kernel Object Manipulation (DKOM) [4]. DKOM’d malware is classified as type II because the Kernel Objects that are used to hide things are normally changed by the OS. This makes DKOM’d malware extremely difficult to detect. Using DKOM techniques, linked lists can be

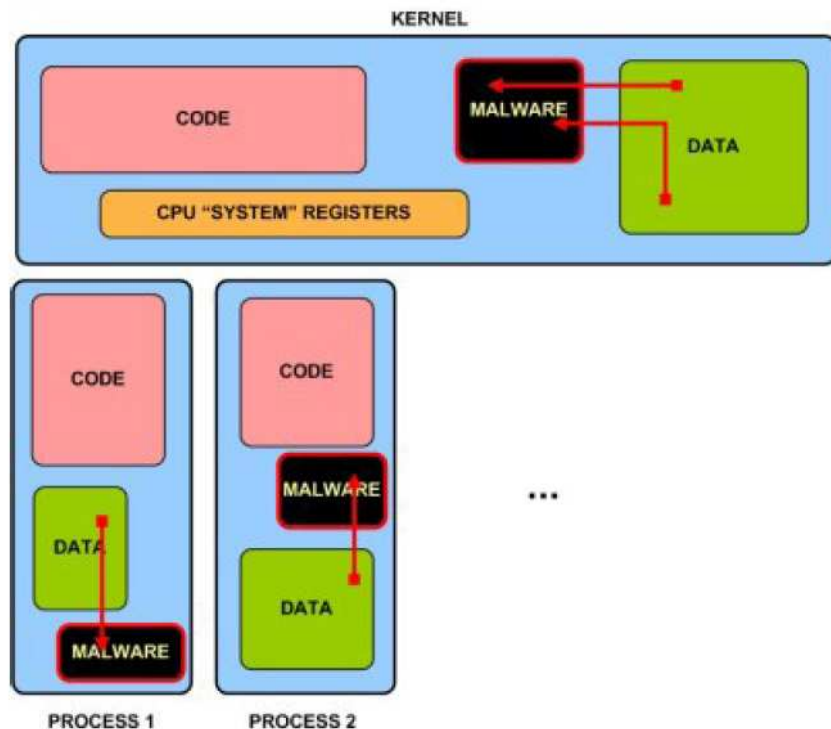


Figure 2.8: Type 2 malware. Subverts OS data [35]

snipped and resewn without affecting their functionality (see Figure 2.9)

2.3.4 Type III Malware. Type III malware is malware that uses hardware virtualization. Virtualization is discussed more in depth in §2.4. Suffice to say for now that type III malware is almost undetectable [35]. Type III malware separates itself completely from the OS. Because the type III malware is ‘closer’ or ‘lower’ to the hardware than the OS there is no way for the OS to have any clue of its presence. There have been a few research groups that have made claims to be able to detect the presence of a virtual machine (VM) running [10,11]. However it has been argued that with virtualized hardware on the rise soon claiming to detect a Virtual Machine Based Rootkit (VMBR) by deducing that a Virtual Machine Monitor (VMM) is akin to claiming you have detected a botnet by observing that there is a network attached. [37] Many have also claimed that in order to negate the affects of a VMBR,

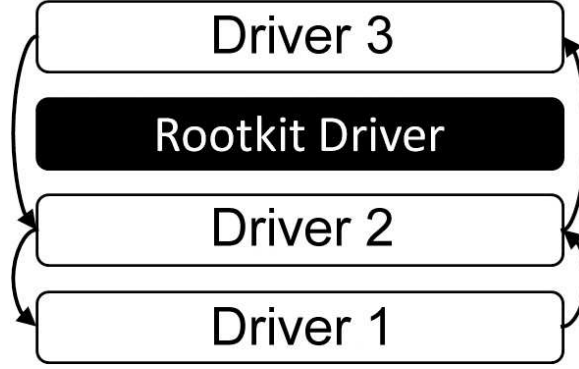


Figure 2.9: Severed and reswenn DKOM’d link list hides rootkit process [29]

one only has to install a trusted verifiable VMM first. This defense was overcome by events when Rutkowska showed that it is possible to nest VMMs [38]. This means that if a VMBR is put underneath a trusted verifiable VMM, that running a trusted VMM could hamper VMBR identification. Any side channel observances could be attributed to the trusted VMM, so defenders will have a harder time deducing if any timing attacks stem from a VMBR.

2.4 Virtualization

Virtualization, specifically a virtual machine based rootkit (see §2.3.4) is a key piece to our methodology of a stealthy Cybercraft initialization sequence. This section explains virtualization in more detail and introduces *Bluepill*, a proof of concept virtual machine based rootkit.

“Virtualization is, at its foundation, a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources [24]”. There are many types of virtualization in practice today, each one abstracting a different component. A non-comprehensive list is given by [24] as: operating system, server, desktop, streaming, storage, data, clustering, grid computing, software-as-a-service and thin client. Each of these cat-

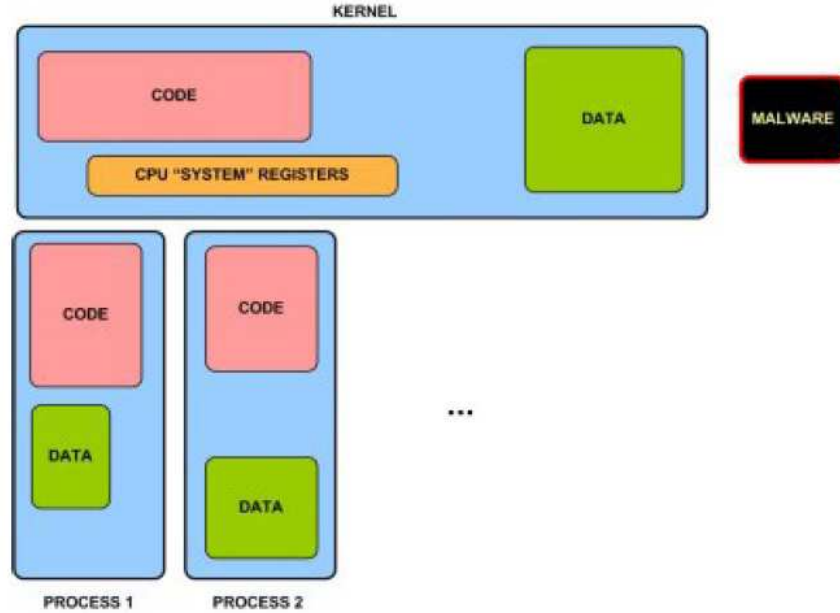


Figure 2.10: Type 3 malware. Is completely separate from OS [35]

egories takes a service or a component and hides the physical characteristics of that component from its users.

There are many benefits to deploying a virtualized system. One of the biggest advantages to virtualization is an increased hardware utilization rate. It was observed by [44] that most servers run at only 15% utilization. That translates to much computer power sitting around being wasted. Under a virtualized system, a virtual machine monitor is placed on one physical *host* computer. The host computer is then installed with many *guest* systems. It was observed by [44] that current IT best practices provision only one application per server. This is because if a server were to fail, only one service or application would be affected. Virtualization provides the capability to marry best IT practices while saving hardware costs, assuming the 15% observation were true and six servers could be run on the hardware that was previously hosting only one. Since VMMs keep the servers in a sandbox, isolated from the other servers, it still maintains the best practice of only one service or application

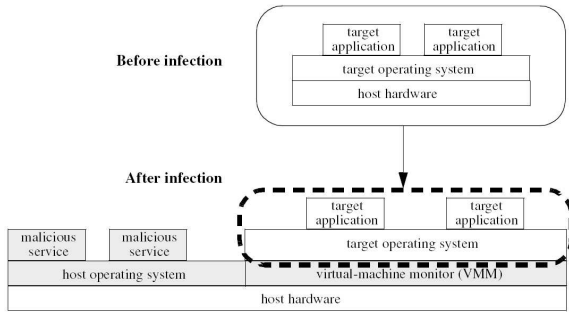


Figure 2.11: SubVirt before and after infestation [20]

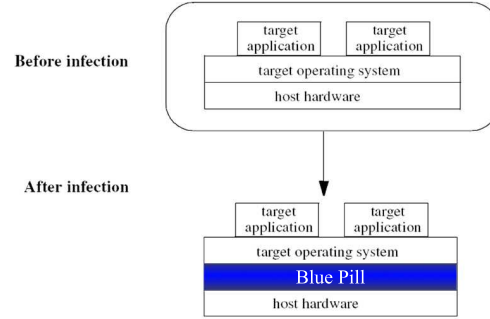


Figure 2.12: Blue Pill before and after infestation

per server. This gives way to substantial cost savings, but not solely in hardware costs.

Historically classical virtualization or full hardware-assisted virtualization (HAV) has not been widely supported or effective on the IA-32 or x86 architecture [1]. Since the x86 is the most prominent hardware architecture in use today [1] new methods to hardware virtualization would need to be explored. The landscape of virtualization changed in 2007 when both Intel and AMD released virtualization extensions to the x86 architecture. Intel calls their extension VT-x and AMD calls theirs Pacifica. Both allow a virtual machine direct hardware access. This permits guest systems direct access to resources that previously caused large delays. For example, guest computers before HAV had great difficulty running three dimensional graphic intense programs due to the software translating that was occurring between the video card drivers and the VMM. Under HAV that bottleneck no longer exists and guest VMs can have direct access to the video card.

2.4.1 Blue Pill. There have been many virtual machine monitors or *hypervisors* written, but none have received the scrutiny and attention as Rutkowska's Blue Pill. [36] In 2006 Rutkowska demonstrated a proof-of-concept VMM that could install on a currently running *host* and transfer that host to a *guest* inside Blue Pill. This is not the first project that married rootkit type delivery mechanisms with virtualiza-

tion. Among the first that completed such a task were researchers at the University of Michigan when they developed the SubVirt VMM. [20] There are many differences that propelled Blue Pill to stardom over SubVirt, first, SubVirt needed to change the boot loader and required a restart of the system, and it was based on a commercial VMM which emulated virtual hardware which would make SubVirt easier to detect (see Figure 2.11). Blue pill on the other hand (see Figure 2.12) overcame all the obstacles facing SubVirt and produced an ultra thin VMM that promised full hardware virtualization and could be installed on the fly and most controversially was touted as “100% undetectable” [36].

2.5 Cybercraft as a Bot

The overall goal of this research is to create a methodology for a Cybercraft to stealthily initialize and plug into a command and control structure. It is important to the Cybercraft project to know if current bot detection techniques and tools will identify Cybercrafts as well. This section serves to point out that bots and Cybercraft do share some common traits. Furthermore, this section describes in more detail how a bot(see §2.2.4) gains command and control.

There are many characteristics that a Cybercraft might share with a common bot. Both a botnet and a Cybercraft need to be controlled by a human. In the case of a botnet thousands of computers are led by a bot herder who will use the botnet for some malicious purpose or for personal gain. A Cybercraft also has the requirement to be human controlled. However, not all Cybercraft need react in the same manner. This separates Cybercraft from a botnet in that Cybercraft are not zombies. In a botnet all bots perform the same action at once. A Cybercraft does not have the same limitation. Additionally, there could be multiple Cybercrafts in the area, and on an execution command they could all perform different actions at different times. But, in both cases it is important for bots and Cybercrafts to establish a command

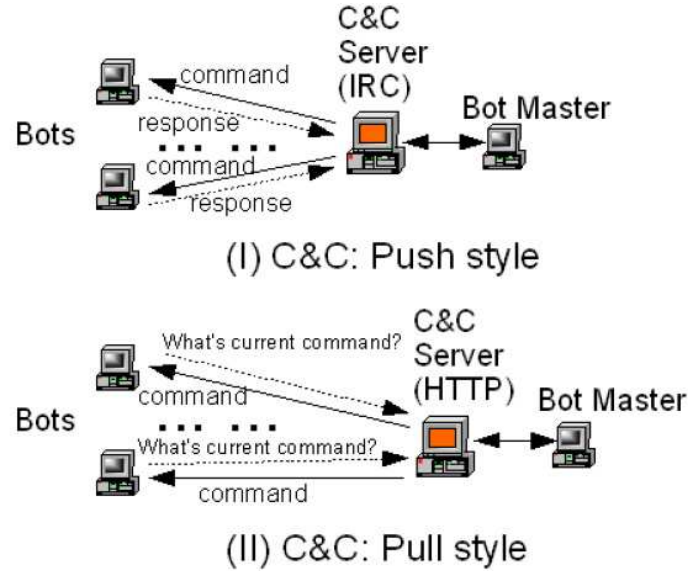


Figure 2.13: Two types of command and control methods [13]

and control channel.

2.5.1 Bot Command and Control. This section details many of the specific characteristics regarding botnet command and control. First, bot command and control mechanisms are usually very stable in that they do not change much once unleashed ‘in the wild’. Second, the command and control architecture of a botnet is regarded by many as the ‘weakest link’ of the botnet [13]. Generally speaking there are two types of bot command and control structure, the push style and the pull style (see Figure 2.13) [13].

Under the push-style construct, a bot-herder or bot master can issue a control command to the entire botnet. Usually this is done simultaneously to all zombies. The zombies then receive the instruction and carry it out. Contrary to the push style is the pull style. Under a push-style command and control structure, the zombies are responsible to ‘check-in’ with the bot master. The bot master would then update the

zombie with any new information or issue a command.

In addition to whether the bot, or a Cybercraft, utilizes a push or pull construct style there still needs to exist some channel or protocol to facilitate the communication. Some of the more common channels currently in use today are: Internet Relay Chat or (IRC)-oriented, Instant Message or (IM)-oriented, web oriented, and channels that rely on their own protocols constructs [18].

IRC is a protocol engineered for synchronous conferencing. IRC allows someone to speak to many people at the same time. This makes it a prime target for botnets. Common IRC botnets require their zombies to connect to a specific IRC server chatroom. The zombie then sits in the chatroom and listen until it gets instructions from the bot master.

IM bots are similar to IRC bots but use common IM channels as communication channels. IM bots are not very popular due to many of the common IM services requiring a user account. Common IM services also are established to make automatic registrations very difficult. IM bots tend to be smaller than IRC bots for the aforementioned reasons.

Web oriented bots commonly use the Hypertext Transfer Protocol (HTTP) to send web traffic to a predefined server. Web bots have many advantages web traffic is common, and as such most Intrusion Detection Systems (IDS) and firewalls will not block the HTTP messages. Web bots are also easier to engineer and implement.

The last major classification of bot communication channels is proprietary protocols. In order for a bot to get information across the Internet cloud, transport layer protocols such as TCP/UDP or ICMP have to be used, otherwise the packets are not

able to be routed and would never leave their local area network.

2.6 *Cybercraft as a Peer-to-Peer (P2P) Network*

One of the aspects of Cybercraft initialization methodology that this work is attempting to demonstrate is Cybercraft's ability, upon initialization, to be able to communicate with neighboring Cybercraft. This methodology requires a Cybercraft to find out what other Cybercraft are nearby, and plug into any command and control channel that is required under the rules of engagement. In order to accomplish this, a Cybercraft must have some way to authenticate messages that it sends and receives. Cybercraft in this respect resembles a P2P or mobile ad-hoc network. A survey has already been completed which analyzed different structured P2P overlay networks against the requirements and specification for Cybercraft [31]. Because Cybercraft can be deployed on different geographic or functional missions it lends itself to a hierarchical peer-to-peer (HP2P) construct.

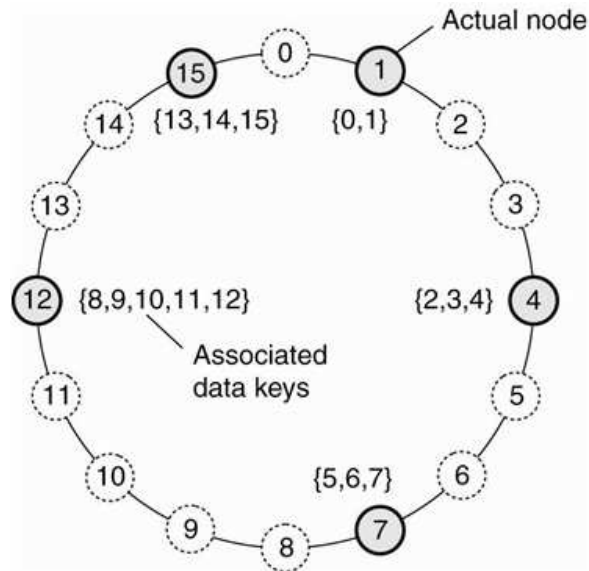


Figure 2.14: The Chord system showing a mapping of items to nodes [41]

2.6.1 Chord. As part of the initialization sequence it is important that a Cybercraft be able to communicate with neighboring Cybercraft. The ability to plug into a command and control network local to the Cybercraft has many advantages. Section 2.1.1 introduced the analogy of a prisoner whispering to other prisoners in an attempt to increase the stealthiness of the communications. A prisoner that can relay all necessary information by listening to a neighboring prisoner is much faster than relying on written letters to family and friends to relay the same information. In addition it is likely that any communication in or out of the prison would be subject to inspection by prison officials. This means that any information regarding something like an escape attempt would be captured and not relayed to the prisoner.

Like the network of prisoners relaying information, Chord presents an system for Cybercrafts to relay information. In order to facilitate a local communications group of Cybercrafts a system must be identified. The Chord system [41] is a good choice for Cybercraft. The Chord system is a distributed hash table (DHT) system that is known to be versatile and scale very well. Under a Chord system each node uses an m -bit identifier to map keys to specific entities. These entities can be anything: files, commands, instructions etc. The Chord system maintains a finger hash table of successor nodes which allows for quick access ($O(\log(N))$ [41]) and relatively low overhead when peers drop in and out of the constellation. Under a Chord system a Cybercraft could join the Chord constellation to find all of the information it needs and stay in or leave the constellation without large disruptions. Figure 2.14 depicts a Chord system with five actual nodes in a four bit (16 max nodes) configuration. Each real node is responsible for the space between it and its predecessor node. This limits the amount of information that has to be shared whenever a new node comes online. Consider the example in Figure 2.14, a new node 10 is to come online. It would only have to transfer information from node 12 (anything associated with keys 8, 9 and 10). If a node is to leave the constellation a node would only have to transfer its information to its successor node. One drawback to this is if a node was to fall

out before it could transfer its information to its successor, the constellation would lose the information the node contained. For this purpose redundancy could be introduced. The issue of redundancy and bad data is discussed in §2.7.1.

2.6.2 Encryption. Knowing that the Cybercraft will be communicating with other Cybercraft brings on additional security concerns. As mentioned in §2.1.2 it is necessary for one Cybercraft to be able to trust other Cybercrafts. Much work has been accomplished in the field of secure multi-party computations (SMC).

“By integrating cryptographic protocols based on secure multi-party computations, software-only protection mechanism can be designed to guarantee the execution integrity and data confidentiality of an agent while it is executed at the remote host.” [27]

The Cybercraft project faces a steep challenge with regards to communication security. It has been shown [27] that there are distinct trade-offs when using secure multi-party computations within such a project as Cybercraft. As the demand for security increases, so does the number of calculations each Cybercraft platform has to make to keep the communications secure. This also increases the number of messages that need to be transmitted to successfully handshake with other cybercraft and conduct any security checks necessary (see §2.1.2). The benefits of encryption are evident however, if a Cybercraft is compromised. Cybercraft compromise in more detail in the next section.

2.7 Cybercraft in the Face of the Enemy

If a Cybercraft is to be able to successfully plug into a command and control channel it must do so even when its enemies might be trying to capture the Cybercraft and reverse engineer its initialization sequence. This section discusses what defensive measures must be considered to be able to initialize when an enemy dis-

rupts the Cybercraft communication network or scrutinizes the Cybercrafts messages.

If captured and analyzed by an enemy, some information could be gained from observing the Cybercraft's messages. If encryption is used, a Cybercraft would only provide evidence that messages are being sent and to whom. No knowledge is gained regarding the information or command and control channel unless the encryption is broken. Of course there are many defensive techniques that could be used to confuse any enemy observers such as dummy messages that are sent so that the enemy could not ascertain which are the true messages and which are the dummy messages, but the enemy would see messages. Thus, part of the Cybercraft stealthiness must rely on the messages not being observed. This is discussed more in §2.9.

Another defensive measure that needs to be considered is that of redundancy. It was introduced in §2.6.1 that a Cybercraft might be taken offline without properly sharing its information to neighboring Cybercraft. This is due to network problems, hardware failure, etc. It is important therefore to incorporate some redundancy into the network so that if a Cybercraft goes offline other dependent Cybercraft don't also lose the required information.

Another problem to consider is that of imposter Cybercrafts. Section 2.1.2 discussed the importance of a Cybercraft trusting other Cybercraft. Imagine a scenario where a Cybercraft is attempting to initialize and plug into a command and control channel only to find that the command and control channel is being operated by the enemy. This problem of Cybercraft reliability and consistency maps to the following Byzantine Generals problem.

2.7.1 Byzantine Generals. In 1982 a landmark paper was written called 'The Byzantine Generals Problem' [22]. The paper outlined a scenario where many

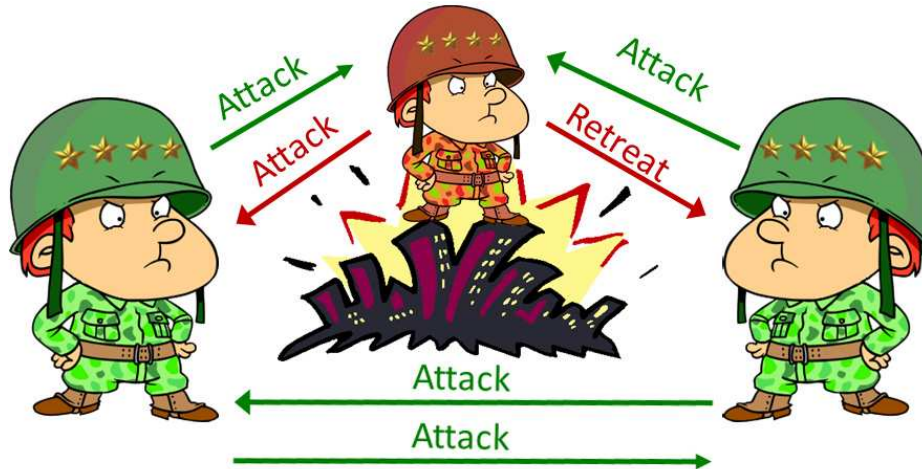


Figure 2.15: Two honest Generals passing verbal messages with one traitor General

divisions of the Byzantine Army were camped out surrounding an enemy city. Each of the divisions were commanded by a Byzantine General. It is the mission of the Generals to observe the actions within the hostile city and develop a course of action. The General's only capability of communicating with each other is via messengers. Of course, those messengers could get captured and replaced with enemy spy agents, or it is possible for a General to go rouge and wish harm upon the Byzantine Army (see §2.1.3 regarding *insider threats*). With the possibility of the sender or the message being compromised it is still vital that all honest Generals agree on a united course of action. For simplicity, the only actions could be to either attack the city or retreat. Both actions are acceptable, however only if all Generals make the same decision. It is a failure if some of the Generals and their divisions retreat while others attack.

This abstract problem was developed to help solve the problems regarding reliability and consistency within distributed systems. The Generals represent processes or nodes in a distributed environment. The pedagogical example of distributed system is the ATM bank machine. If a bank customer attempted to withdraw money from his account, but there was a server failure somewhere down the chain, how will the failure get recognized? How do you prevent the system from withdrawing from a user's account, but not give them the money? The Byzantine General paper showed

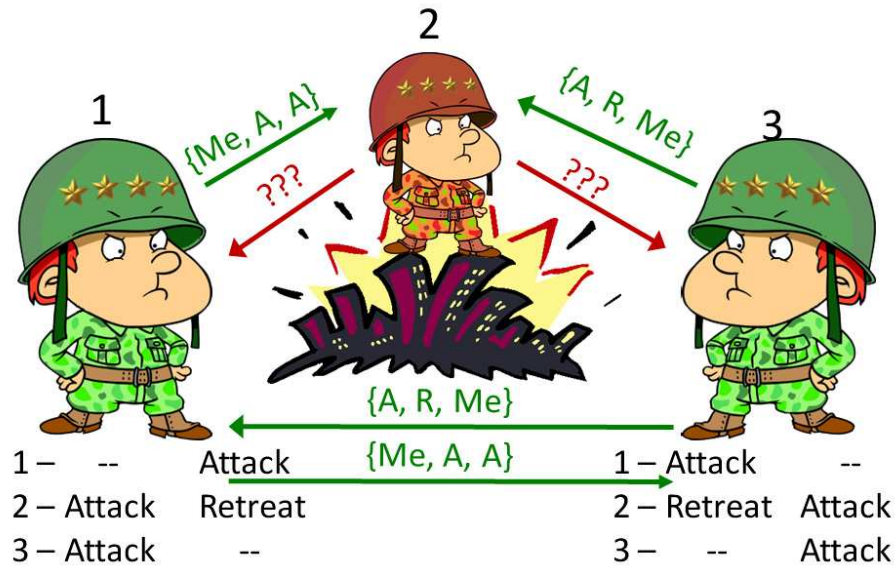


Figure 2.16: Two honest Generals passing signed written messages with one traitor General

that when using simple messages for every traitorous General m you would need $3m + 1$ loyal Generals if the loyal Generals are to be successful. Notice on Figure 2.15 that with less than $3m + 1$ loyal Generals they are not all in agreement, and cannot deduce which General is the traitor.

Suppose now that the Generals were to improve their communications protocols. Suppose that instead of verbal messages that the Generals were required to sign a separate letter to each other General with their vote on what action should be performed. Then after a General has received all of the letters from all the other Generals they are put together. At this point each General would have a local list of all of the votes that all the other Generals have given to just them. This is shown as the leftmost columns in Figure 2.16. After all of the first round of messages have been tallied the Generals are to send out another letter to all of the other Generals with a note detailing all of the votes that General has received from the other Generals. This is depicted by the green sets going across the green lines of Figure 2.16. Note, the

traitor is not following protocol, he is sending garbage rather than the requisite letters.

At the cost of a second round of letters, and the time it takes to process all of the letters all honest Generals will not only be able to make the correct decision, but they are also able to identify who all of the traitorous Generals are in the mix. With signed messages you can deal with any number of m traitors, regardless of how many honest Generals exist.

If a Cybercraft is to not fall victim to an impostor, it needs to be in communication with $3m + 1$ loyal Cybercrafts for every m impostors, or use an implementation where signed messages could be used. Of course using signed messages increases the number of message traffic that would need to be sent and has a detrimental effect on the stealthiness of the Cybercraft initialization sequence.

2.8 Bot Hunter

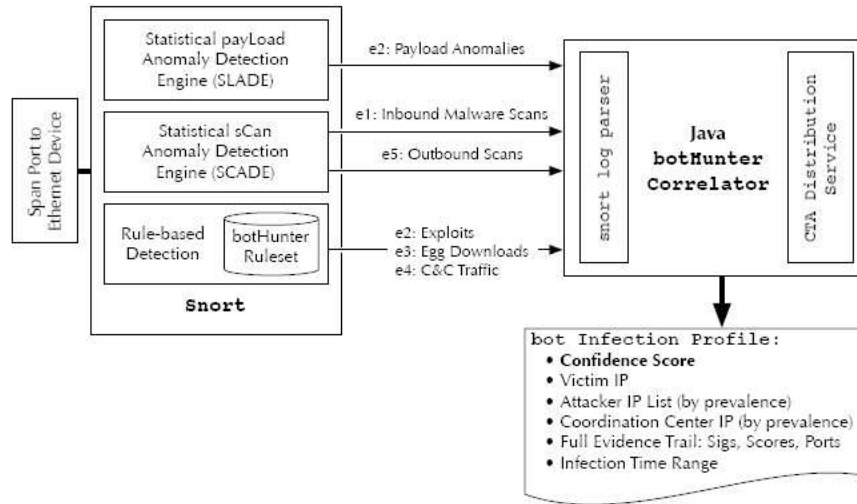


Figure 2.17: BotHunter Architecture [12]

Bothunter is a bot detection utility that claims to be “...the first real-time analysis system that can automatically derive a profile of the entire bot detection process,

including the identification of the victim, the infection agent, the source of the egg download, and the command and control center” [12].

Bothhunter is a trio of IDS components that attempts to correlate data from watching inbound and outbound information. Figure 2.17 shows the architecture of Bothhunter. A large portion of Bothhunter is similar to any other rule based IDS. Bothhunter is special in the fact that it uses two other IDS components that serve to monitor the entire network traffic and make observances that are special to botnets. For example in a botnet when an order is given all bots immediately respond. If there are five bots in one network and Bothhunter notices that they all similarly react in the same way at the same time it can put those pieces together and create a profile that alerts a network administrator or a forensics examiner to the bot.

Bothhunter was released for public distribution in November 2008. Its authors have made claims that BotHunter represents the most in-depth network-based malware infection diagnosis system available today [17]. Section 2.5 drew the comparison of Cybercraft and botnets sharing similar characteristics. This research uses Bothunter to see if a Cybercraft initializing and joining a command and control channel will be flagged as a botnet. Furthermore in Chapter 3 and 4 Bothhunter is put to a series of tests to assess many of the claims that its authors are making.

2.9 Needle in a Haystack

One of the ways that the stealth of a Cybercraft is measured is by evading detection of such utilities as Bothhunter. Another technique to improve the stealthiness of Cybercraft messages is to hide in plain sight. One of the main characteristics that differentiate a good rootkit from a bad one is the concept of stealth (see §2.2.3). In [14], the authors point out that one of the most successful ways to remain stealthy is to “hide in plain sight”. One of the reasons that looking for a needle in a haystack is a daunting task is because a needle looks a lot like a straw of hay, even though in

actuality a needle is very different from a straw of hay.

If one was to examine the traffic of a network there are many protocols that are common. Internet traffic can almost always be found in the form of HTTP messages on port 80 for example. This is the concept behind many firewalls and intrusion detection systems. They exist to deny things that are different, or in other words, should not be there. If a malicious packet of information looks similar to a legitimate one it would be allowed to pass through the defenses (see §2.1.2). The only other defense is relying on the skill of a network administrator or forensics examiner to know the network well enough to spot malicious traffic through examination of network logs.

The authors of [23] admit that examination of network logs or “network forensics” is a challenge. Problems include the sheer volume of data as well as the number of protocols that could be encountered. In addition, it is a daunting task to stay ahead of the network activity of popular software applications.

The goal of this research is to create a stealthy Cybercraft initialization sequence. The methodology is based on hiding in plain sight and evading automatic utilities such as Bothunter. After all the automatic tools have sniffed the network traffic, in the end, the stealthiness of this research maps to a human searching through network logs. This is understood to be an arduous task and a hard problem.

2.9.0.1 Address Resolution Protocol. ARP packets are critical to the methodology for a Cybercraft to stealthily initialize and plug into a command and control channel. ARP is a special protocol that is used to resolve a computers IP from its MAC address. The key components of a ARP packet are the source IP address, the source MAC address, the destination IP address and the destination MAC address. Depending what information is trying to be ascertained, certain fields will be used

and others could be left blank.

For example, imagine the scenario where a computer is trying to send a packet of information to IP 1.2.3.4 but it does not know the MAC address of the recipient. In order to find out what MAC address is associated with IP address 1.2.3.4, a broadcast is issued to all of the devices that share a network with the computer. The message is basically if you are 1.2.3.4 send your MAC address to the requester. Of course not all fields are relevant. If I am asking for your MAC address then it is not necessary to put anything in the Destination MAC address field. Any information placed inside the Destination MAC address field normally will not even be looked at. Filling the destination MAC address on an ARP request would be like telling a device their own address, which if known would remove the need to send the request in the first place. Of course there is not a penalty for using that space for other reasons. ARP requests are a very common part of a network. Computers do not usually keep ARP caches for a long time and are constantly probing their neighbors for addresses. These benefits can be leveraged to communicate more than just addresses. If a code was placed in the unused fields of an ARP request and inversely an ARP reply devices on a shared LAN could communicate back and forth. This research will use these modified ARP messages to stealthily communicate command and control information between Cybercraft platforms.

III. Experimentation

This chapter outlines our research methodology for testing the stealthiness of an initialization sequence for a Cybercraft platform. All assumptions regarding the framework of the methodology used are given and our research hypothesis and goals are presented. Three scenarios are introduced which serve to simulate common situations in which a Cybercraft platform may likely operate. An explanation is given regarding how the initialization sequence operates under each of the three different scenarios. The scenarios are simulated in a controlled environment. The results of the experiments are found in Chapter four.

3.1 *Problem Definition*

The principal aim of this research is to *create a stealthy initialization sequence methodology for the Cybercraft platform to plug into a command and control channel*. §2.1 explains the basics of Cybercraft while §2.5 relates the similarities of a Cybercraft and a botnet. Specifically, the similarities that both require plugging into a command and control channel. §2.8 details *Bothunter*, a current tool that claims to be capable of detecting botnets and their associated command and control in real time [12]. This research examines the claim made by Bothunter that it can in fact detect botnets in real time. Additionally, Bothunter is used to gauge the stealthiness of the proposed Cybercraft initialization sequence.

3.2 *Hypothesis and Goals*

This research attempts to satisfy the following goals:

- Cast the Cybercraft C2 problem as a botnet detection problem.
- Create a methodology for a Cybercraft platform to stealthily initialize and join a command and control channel

- Provide an evaluation of Bothhunter based on our analysis
- Test that the aforementioned initialization sequence will not be detected by the botnet detection tool Bothhunter

The first contribution of our research is to cast the Cybercraft C2 problem as a botnet detection problem. As discussed in §2.5 Cybercraft platforms share multiple characteristics with botnets. The main goal of this research project is to create a stealthy initialization sequence for a Cybercraft platform to plug into a command and control channel. This goal directly contributes to the Cybercraft project. It is a requirement for a Cybercraft to be able to initialize and communicate in a command and control channel. This research sets out to answer that requirement, which will not only be of value to the Cybercraft project, but to the U.S. Air Force’s mission to fight and win in Cyberspace [9]. It is our hypothesis that an initialization sequence that plugs a Cybercraft platform into a command and control channel is not only possible, but it can be shown to be stealthy with regards to current botnet detection tools.

Another goal of this project is to test the claims made by the authors of Bothhunter. §2.8 explains in greater detail what Bothhunter is and how it works. In order to make any claims as to the stealthiness of the proposed Cybercraft initialization process it must first be demonstrated that Bothhunter is in fact competent in detecting botnets. Our definition of stealthiness can be found in §3.3.1. Bothhunter was born in the academic community, and this research is of value to that same academic community by serving as an additional unbiased voice as to the validity of the claims made by the authors of Bothhunter [12]. It is our hypothesis that Bothhunter works as advertised and is successful in detecting a number of Botnets in real time.

The final goal of our research is to show that our proposed initialization sequence for Cybercraft will not be detected by Bothhunter. It is intended that the initialization sequence be a stealthy one. For the purposes of this research, the initialization

sequence was considered stealthy if common stealthy techniques were employed and it was not detected by the real time botnet detector Bothunter. It is our hypothesis that our initialization sequence incorporates many stealth techniques learned from the malicious software community 2.2 and will be capable of evading Bothunter.

3.3 Setup

In order to successfully achieve the goals of this research we have created a controlled research environment. The purpose of this environment was to simulate, as best as possible, the real world; but, because we used known malicious botnets, the environment was completely isolated from any other network. As shown in Figure 3.1, we have created two networks. The first network (10.1.30.x) modeled a trusted inside network as could be found in an office. The second network (10.1.90.x) modeled an outside untrusted network which could represent someone plugged into the Internet. For the sake of convenience we created this research environment within VMware Workstation 6 running over commodity hardware running Windows XP Professional SP3. The inside network contained five workstations running Windows XP Professional SP2. The outside network only required one workstation to simulate an untrusted attacker outside the network and also runs Windows XP Professional SP2 (see Figure 3.1).

In order to setup the inside network, first we installed Bothunter, and on the outside network we installed UnrealIRCd an IRC server and Easy Web Server an Apache web server. The IRC server was set up for botnet connections. The web server was set up so simulated users inside the network have real webpages to download. The purpose of using real webpages was to pass real HTTP traffic back and forth on the network and under the nose of the botnet detector. In addition to the aforementioned software we installed Wireshark, a network protocol and packet analyzer tool, on all of the inside and outside workstations. All of the workstations

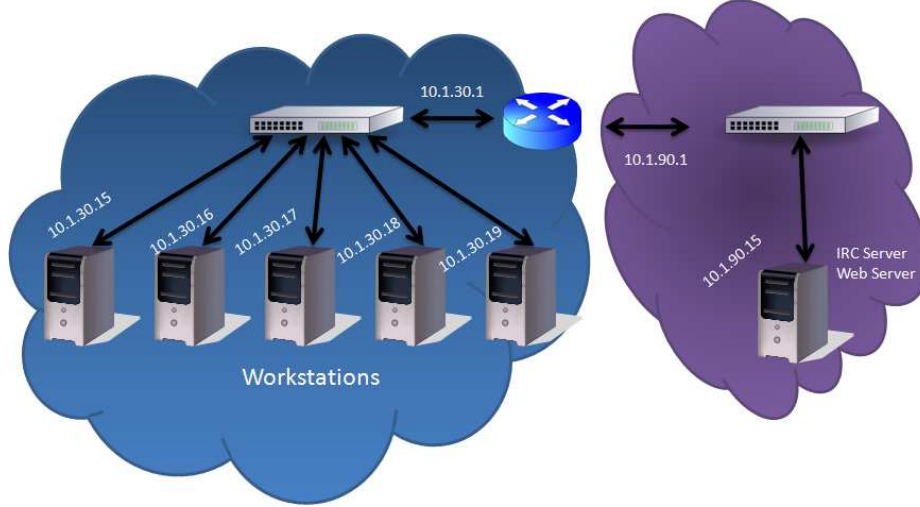


Figure 3.1: Proposed network for experiment

have mIRC, an IRC client application, installed. MIRC allowed us to connect to the UnrealIRCd server and send IRC messages back and forth. Finally we installed Packet Builder on (10.1.30.16) of the inside workstations. Packet Builder allows us to build any kind of TCP, UDP or ARP packet that we desire and send it on the network.

3.3.1 Measuring Stealth. The main focus of this research was to create a stealthy methodology for a Cybercraft initialization sequence. This research casts our methodology as a bot detection problem, and as such measures the stealthiness of the methodology based on state of the art Bothunter detection utility.

We understand that evading a bot detection tool does not completely guarantee that the methodology is stealthy. The field of computer forensics is filled with different methods for identifying the existence of programs that are designed to be stealthy.

Although many utilities and methods exist, we feel that because a Cybercraft platform shares many similarities to botnets, using a bot detection tool is a good measure of stealthiness. Bothunter was chosen as the bot detection tool because it

is lauded as being the most advanced bot detection tool, and the first one that can detect bots in real time [12].

3.4 Approach

The first step needed for our experimentation is to ascertain if the bot detection tool Bothunter fulfills its claims to be able to detect botnets in real time. To this end, we conducted experiments similar to the ones first conducted by the authors of Bothunter and can be found in [12]. In their experiments they tested 10 different bots. For our purposes we only tested Bothunter against four bots. Three of the bots were used in the original testing and one (Sdbot) was not tested against Bothunter in the original paper. See Table 3.1 for a list of bot names and versions used during this research.

Table 3.1: The name and version of the bots chosen to use in experiment

	Bot Name	Bot Version
1	Agobot	3-priv4
2	Phatbot	alpha1
3	Rxbot	7.5
4	Sdbot	05b

In order for the botnets to work properly on our closed research network it was necessary to recompile all of the bots specifying the command and control IRC server and channel that were used. Since we do not have a working DNS server on the closed research network, we hardcoded the IRC servers IP address. The external workstation used to simulate a malicious bot herder and host the IRC server is 10.1.90.15 (see Figure 3.1). Once compiled we introduced the bots onto four of the five internal workstations. Next we were able to ‘herd’ them through the IRC command and control channel.

Once the bots are functional on the research network, the next step conducted was to gauge the effectiveness of Bothhunter by running a series of tests. Each test lasted 90 minutes in length, and the number of malicious profiles that Bothhunter generated were recorded. Figure 3.2 shows a sample screenshot of Bothhunter from [17] with malicious profiles generated identifying the existence of botnets.

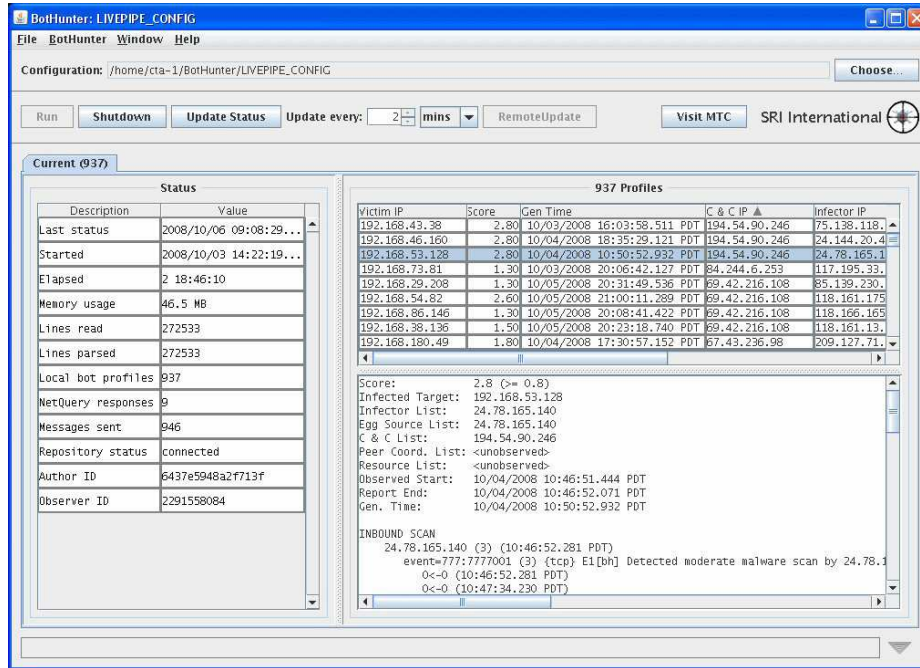


Figure 3.2: Screen shot from Bothhunter [17]

We identified twelve tests designed to demonstrate the effectiveness of Bothhunter (see table 3.2). The first test is strictly a control to see if Bothhunter announced any ‘false positives’. Tests 2-5 identified if Bothhunter could detect the respective bots. Tests 6 & 7 were designed to see if Bothhunter is affected by network traffic volume. Tests 8 - 10 were to gauge how the level of scanning affected Bothhunters detection rate, and finally tests 11 & 12 were designed to see how Bothhunter handled bots that were scanning in addition to increased levels of network traffic. The traffic used was captured traffic from an exercise network during AFIT’s participation in a Cyber Defense Exercise. For the exercise a complete network was set up that

simulated the functions that a real operational network would require such as email, DNS, a webserver, instant messaging (chat), active directory. The exercise network IP address range was 10.1.30.1 /24. This is the same network mask as our research network. This was done so that all additional traffic introduced appeared to Bothhunter as originating from the inside network. The additional traffic was introduced via the software program *PReplay 1.1*, a tool designed to replay captured packets.

Table 3.2: A brief description of the series of tests performed to ascertain Bothhunter effectiveness

#	Test Discription	#	Test Discription
1	Network only	7	Agobot with heavy traffic
2	Agobot only	8	All bots, no scanning
3	Phatbot only	9	All Agobot scans, conservative
4	Rxbot only	10	All Agobot scans, aggressive
5	Sdbot only	11	All Agobot scans, conservative, light traffic
6	Agobot with light traffic	12	All Agobot scans, conservative, heavy traffic

There are two main criteria that we looked for during the twelve tests.

- If Bothhunter identified the bot
- If Bothhunter identified the command and control channel

These two criteria are particularly important to our research. After the completion of each 90 minute test, we imported the data into a condensed table (see tables 3.3, 3.4).

3.4.1 Assumptions. The next step to realize our methodology of a stealthy Cybercraft initialization sequence was to simulate the sequence itself. There are a number of assumptions that we have made with regards to the status of the Cybercraft platform. This section serves to explain the assumptions we have made.

The purpose of this research project was to develop a methodology for stealthy Cybercraft initialization. It was not to build a Cybercraft platform. With that being

Table 3.3: Showing bot detection as first criteria

Test Description	# of Profiles Detected
Network only	
Agobot only	
Phatbot only	
Rxbot only	
Sdbot only	
Agobot with light traffic	
Agobot with heavy traffic	
All bots, no scanning	
All bots, light scanning	
All bots, heavy scanning	
All bots, light scanning, light traffic	
All bots, light scanning, heavy traffic	

Table 3.4: Showing C2 channel detection as second criteria

Test Description	C2 Channel Detected
Network only	N/A
Agobot only	
Phatbot only	
Rxbot only	
Sdbot only	
Agobot with light traffic	
Agobot with heavy traffic	
All bots, no scanning	
All bots, light scanning	
All bots, heavy scanning	
All bots, light scanning, light traffic	
All bots, light scanning, heavy traffic	

considered it was assumed that a Cybercraft platform exists. In order to increase the Cybercraft platforms stealthiness it was assumed that the platform is modeled after the virtual machine based rootkits that are discussed in §2.3.4 & §2.4.1.

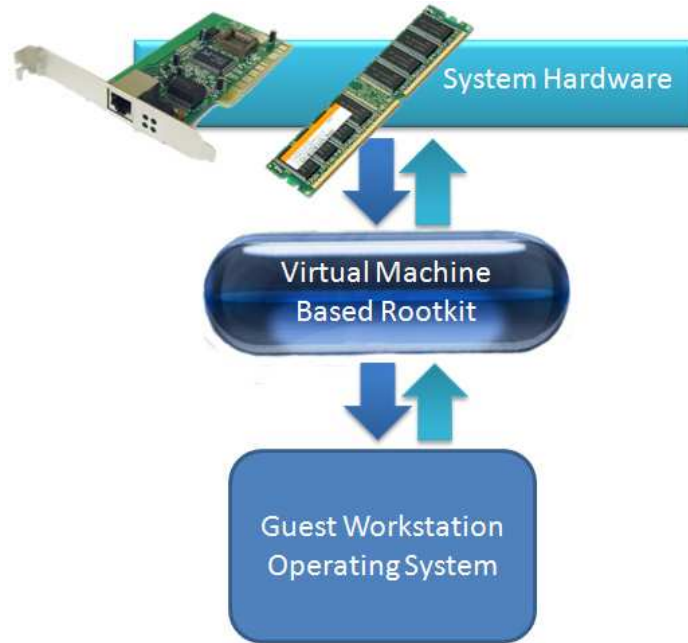


Figure 3.3: Showing that a Virtual Machine Based Rootkit is closer to the hardware than the Guest Operating System

By using a VMBR the client is not able to see the messages generated by the Cybercraft initialization sequence. We assumed that the rootkit, which would be closer to the network interface card than the workstation operating system (see Figure 3.3), would be capable of hiding any evidence of the Cybercraft's existence from any user of the workstation itself. This greatly reduces the chances of Cybercraft platform detection, and leaves the open communication as the greatest source of exposure. As noted in §2.9, finding malicious traffic while sniffing all of the traffic can be a daunting task. We assumed that this initialization sequence takes place on an operational network that is allowing and using ARP, HTTP and DNS protocols. If this initialization sequence were to be executed in a network where the aforementioned protocols were not allowed, the Cybercraft platform would stand out and risk exposure.

Another assumption was that the rootkit is already planted on the guest workstations. It is not the intent of this research to develop effective infection or installation techniques. A summary of many common installation vectors can be found in §2.2. Also, as previously mentioned it is beyond the scope of this research project to develop a working Cybercraft platform. Instead, it was assumed that the Cybercraft platform follows the pseudocode outlined in the Cybercraft initialization sequence (see Algorithm 1). In order to not risk unnecessary exposure, the initialization sequence first tried to contact other Cybercraft on the same network as this is a quieter operation (see §2.6.1). After looking locally, the Cybercraft attempted to retrieve information using DNS messages. If unsuccessful the Cybercraft attempted to reach a predetermined webpage for information. Each of these scenarios are broken down and explained in greater detail in the following sections (see §3.5.1, §3.5.2, §3.5.3).

Algorithm 1 Pseudocode for Cybercraft Initialization Sequence

```

1: while Cybercraft Terminate Order Not Received do
2:   Choose a random number of seconds  $\rightarrow S_1$ 
3:   Listen  $S_1$  seconds to network traffic for neighboring Cybercraft
4:   if Neighboring Cybercraft are found then
5:     Note Cybercraft IP and MAC address
6:     Wait some random amount of time  $\rightarrow S_2$ 
7:     Contact neighboring Cybercraft
8:   else
9:     Send a modified DNS message
10:  end if
11:  if Special DNS message is returned then
12:    Request C2 information from DNS reply
13:  else
14:    Send a HTTP request to a hardcoded address
15:  end if
16:  if Special HTTP reply is returned then
17:    Request C2 information from HTTP reply
18:  end if
19: end while

```

Another assumption made is that the Cybercraft platform would be capable of joining a Chord type system as explained in §2.6.1. With the capability of joining a

P2P local network the Cybercraft is able to communicate with other Cybercraft and be able to plug into a command and control network locally (see Figure 3.4). Local communications is preferred over sending communications outside the network, which might be scrutinized closer by a firewall, IDS or router and increases the potential of alerting a network administrator to the presence of the Cybercraft.

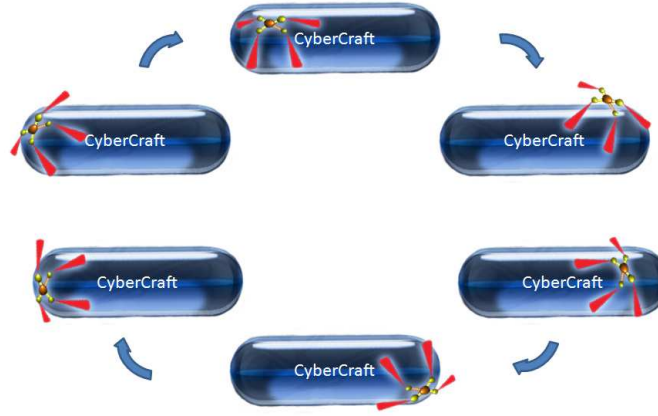


Figure 3.4: CyberCraft displayed as a Chord Ring of Virtual Machine Monitors communicating with each other

The final assumption is that the Cybercraft initialization sequence works the same in a real world network with separate pieces of commodity hardware as it performs in the VMware closed research network. Since VMware gives each virtual machine access to actual hardware, it is assumed that the initialization sequence would act the same in the real world and that any detection made by Bothunter on the research network would be made in the real world and vice versa.

3.5 Scenarios

The next step of the research is to test the Cybercraft initialization sequence against Bothunter and see if Bothunter read the Cybercraft platform network traffic and identified the traffic as a botnet. Three scenarios are introduced that were sim-

ulated on the research network with Bothhunter running. Each scenario has different degrees of stealthiness and conversely different degrees of risk of Cybercraft compromise and are explained in the next three subsections.

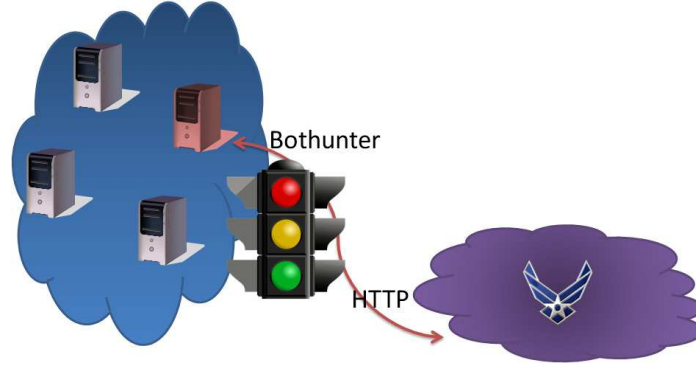


Figure 3.5: First scenario, only connection is using HTTP

3.5.1 Scenario I. The first scenario assumed initializing a Cybercraft when there are no other Cybercraft on the same local network. Bothhunter was running on this network and, by using a hub, has full visibility of all network traffic. This scenario requires the Cybercraft platform to attempt to connect to a webserver and download a webpage (see Figure 3.5). It was assumed that the webpage addresses has been hardcoded into the Cybercraft platform. For stealthiness many webpages could be used, including decoy web pages. In this exchange the Cybercraft platform requests a particular webpage that was identified ahead of time to signal to the webserver that a Cybercraft platform is attempting to plug into a command and control network or channel. Once received, the web server could create a webpage specific for that Cybercraft with encrypted instructions inside the HTTP replies. Of the three scenarios, we hypothesized that this scenario is the least likely to succeed. I believe this because the method of communicating resembles other botnets (see §2.5.1). The HTTP traffic generated by the Cybercraft platform were regular HTTP GET requests. Of course, in practice stenography could be used to further hide Cybercraft actions from any malicious observers, sniffers or network administrators.

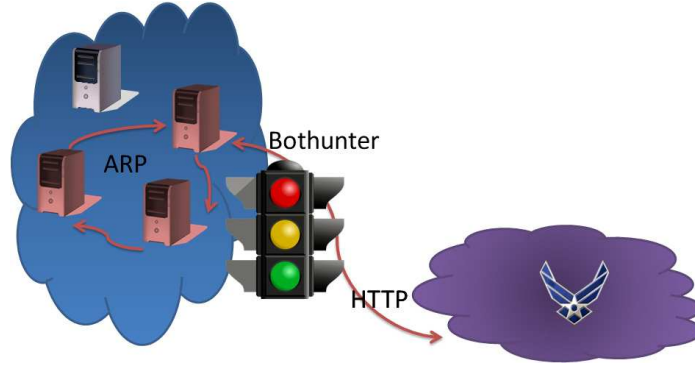


Figure 3.6: Second scenario, Cybercraft gains plugs into local Chord Structure using modified ARP messages

3.5.2 Scenario II. The goal of the second scenario is to simulate initializing a Cybercraft on a network that has other Cybercraft operating on it (see Figure 3.6). This scenario assumes that at least one of the preexisting Cybercraft is plugged into a command and control channel using HTTP (see Scenario 1 §3.5.1). In this scenario the Cybercraft listens for other Cybercraft via modified ARP messages. ARP was chosen because it is small and quick and available to everybody on the same LAN. After a random amount of time the Cybercraft attempted to communicate with one of the neighboring Cybercraft and request to join a Chord based communications system. The traffic for the following steps was created:

- New Cybercraft contacts established Cybercraft
- A handshake occurs
- New Cybercraft receives instructions on its place in the Chord architecture
- New Cybercraft contacts Chord successor and transfers information

These steps simulated the steps discussed in §2.6.1. In order to reduce the risk of compromise all data was assumed encrypted as per §2.6.2. The second step of a handshake was necessary to be able to independently validate each other as per §2.1.2. For the purposes of this research the handshake was limited to a 96-bit request message, a 48-bit response message and a 480-bit information transfer. Once fully connected to a local Chord structure, it was assumed that a Cybercraft can send

and receive command and control instructions, first locally, then via the tethered connection to the outside.

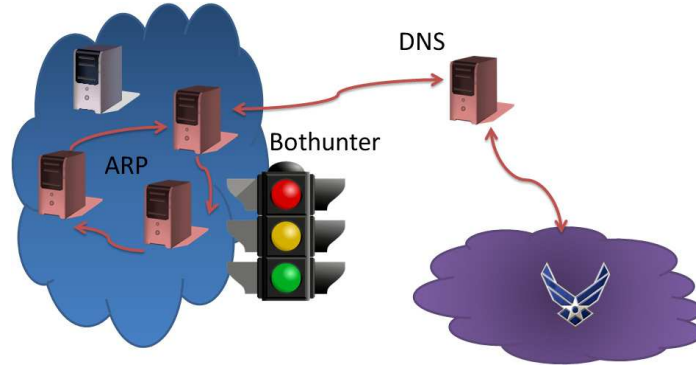


Figure 3.7: Thrid scenario, Cybercraft platform connection to outside C2 structure via DNS messages

3.5.3 Scenario III. The final scenario assumed the same Chord structure exists similar to scenario 2 (see §3.5.2). This scenario also assumes that a Cybercraft was in place somewhere outside the network boundary, where it could sniff DNS requests. We also assumed that this Cybercraft is outside of the perimeter and has unfettered access to a command and control channel outside the local network (see Figure 3.7). This scenario was to test if sending DNS requests would trip the botnet detectors. This scenario should arouse less suspicion on the part of a botnet detector as DNS is a much less common command and control channel than HTTP traffic.

3.6 Summary

This chapter explained our research methodology for testing a stealthy initialization sequence for the Cybercraft platform. It detailed the hardware and software that we used as a test network as well as outlined the pseudocode for the Cybercraft initialization sequence. This chapter listed all the assumptions made to complete this research. Finally, it detailed the experiments we conducted to test if Bothunter was capable of detecting bots in real time, as well as outline the Cybercraft initialization scenarios that we tested against Bothunter.

IV. Results and Analysis

This chapter gives the detailed results of our experimentations that were outlined in Chapter 3. This chapter explains the results of testing Bothunter against known bots. It also explains the results of testing a Cybercraft initialization sequence against Bothunter. The significance of the findings is presented. Another experiment was presented that could work in parallel to bot detection to automatically detect a Cybercraft when using our presented methodology.

4.1 *Testing Bothunter*

The purpose of this research effort was to identify a stealthy initialization sequence for a Cybercraft to plug into a command and control channel. It was presented in §2.5 that the characteristics of a Cybercraft could expose itself to botnet detectors. Bothunter was selected to test if it would detect the Cybercraft when using the initialization sequence presented in §3.4.1. However, before any claims were made as to the stealthiness of the Cybercraft initialization sequence Bothunter must first be tested to ascertain that it in fact is capable of detecting bots in real time.

In their paper [12], the authors of Bothunter used their bot detection tool against Agobot3-priv4, Phatbot alpha1 and Rxbot 7.5. We acquired these bots along with Sdbot 05b to perform our own tests against Bothunter. The first step that was performed was to compile the four bots with the IP address of our IRC server. All four bots use IRC as their means of establishing a command and control channel. Of the four bots SDbot was the most straightforward to compile. We originally ran into difficulty compiling Agobot and Phatbot. They were written using single threaded libraries, the compiler we were originally working with would not build the file correctly. This setback was easily fixed by reverting to an older compiler. All four bots required us to identify the host address of the IRC server to be used by the bot herder. In addition to the IP address we had to establish the channel to serve as the

command and control channel, and establish a password to log into the bot. Once the research network was installed and functional as described in §3.3 It was just a matter of executing the files. Once executed the bots would contact the IRC server and join the predetermined channel. All we had to do was open an IRC client application, join the same channel and log into the bots using the password specified in the compiled code.

It is beyond the scope of this research project to go into further detail on the capabilities of each of the four bots. However, we do want to point out a few differences that are key to some of the observations that we have made during our testing. First, Agobot, Phatbot and Rxbot all have scanning capabilities built into them. This is used to scan for vulnerabilities to be exploited. Sdbot 05b does not have a scanning function built into it. Additionally, Rxbot uses a home grown cryptography system to encode many of the commands it uses. It does have an option that allows it to work in the clear without the cryptography portion. Throughout the source code, comments are made advising against not using the cryptography. We found that Rxbot without using the cryptography portion was inconsistent. We think there are bugs in the bot source code when not using the cryptography. Because we could not get consistent results from Rxbot, we dismissed this from any further tests.

4.1.1 Bothhunter Experiments. Table 4.1 shows at a glance the number of profiles that Bothhunter detected during each one of the tests. The rest of this section explains in further detail the tests performed and the results observed.

The first test was performed in order to create a baseline of the research network traffic and to verify if Bothhunter is prone to false positives. One of the metrics that we kept track of was the number of packets that traversed the network during each of the 90 minute tests. Additionally, we saved each packet from each one of the tests

Table 4.1: Table showing bot detection results

Test Number	Test Description	# of Profiles Detected	# of Network Packets
1	Network only	0	1,490
2	Agobot only	204	4,115
3	Phatbot only	24	98,001
4	Sdbot only	0	2,533
5	Agobot with light traffic	131	74,840
6	Agobot with heavy traffic	99	529,598
7	All bots, no scanning	0	4,889
8	All Agobot scans, conservative	73	34,612
9	All Agobot scans, aggressive	73	33,990
10	All Agobot scans, conservative, light traffic	73	74,840
11	All Agobot scans, conservative, heavy traffic	73	468,785
12	Constant Sdbot UDP flood	0	93,757

in case there was a situation where we desired to replay the test. We found that the total packet number from each test gives a fair indication as to the noise level of each test. For the first test there were no bots on the network. With no bots, the network produced 1,490 packets of traffic. Bothunter produced no false positives.

The second test was with Agobot on the network. Agobots default settings are very ‘noisy’ relative to the quiet network. The network with Agobot produced 4115 packets of traffic, which is 36% more traffic than the network alone. Bothunter had no problem whatsoever in identifying Agobot on an otherwise quiet network. Bothuner reported 204 profiles at the end of this test. This is by far the greatest number of profiles generated.

The third test was with Phatbot alone on the network. Phatbot is a newer version of Agobot and has more command options associated with it. This scan caused great difficulty. When given the command to start the scan, Phatbot monopolized the CPU. With five workstations running and driving a constant 50 - 60% of the virtual machine CPU it 100% maxed out the actual processors. As a result Wireshark the

network capture utility was bogged down as was the keyboard, mouse and all other software. It is unknown if 98,001 network packets is the true number or how the slowdown affected Bothunter.

The fourth test was with Sdbot alone. As previously mentioned Sdbot does not scan the network looking for open ports or other associated vulnerabilities. Sdbot does send out keep alive messages, and over the 90 minute period had produced 2,533 packets on the network. Bothunter did not generate any profile for Sdbot. Sdbot was configured to use the default IRC ports and maintained it's command and control channel throughout the test. After the 90 minute period expired and the results were captured, we attempted to execute all of Sdbots commands. This included a DDoS attack on one of the five workstations. We got constant status updates and computer and network information from the workstations. Finally, we set up a file server on the external workstation and used Sdbot to command all infected workstations to download and execute an html file that we ironically named *innocent.html* (see Figure 4.1). Despite all the malicious logic and constant stream of IRC command and control traffic Bothunter failed to generate a profile.

The fifth test was devised to see if Bothunter would be as successful in identifying botnets in the presence of a noisier network. In other words, §2.9 points out the problems associated with finding a 'needle in a haystack'. This test, along with the sixth test were designed to build a bigger 'haystack' for Bothunter. During this test a small amount of traffic was introduced. As identified in §3.4 the traffic was taken during a Cyber Exercise and contained traffic associated with a busier network than our research network. This test introduced 74,840 packets of traffic on the network. The results of this test support our hypothesis that in a noisy network it is harder to spot the malicious packets. Even though nothing changed with the bot, and Agobot performed the same scans it did during the previous test, Bothunter only identified

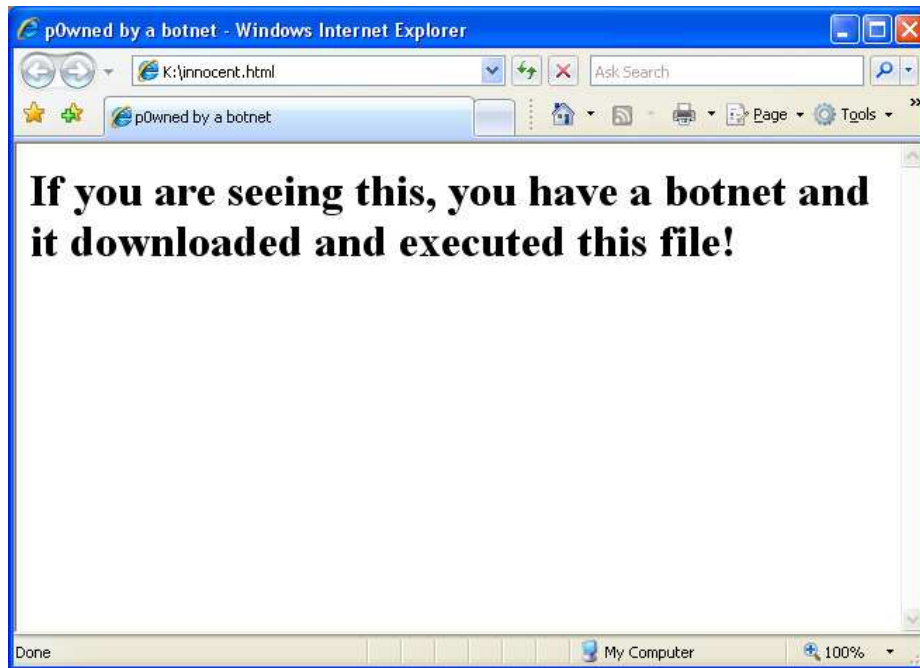


Figure 4.1: Screenshot of a file that was downloaded and executed using Sdbot

131 profiles. This is a 64% drop in detection from Agobot on a quiet network.

The sixth test introduced even more traffic on the network. For this test I used Sdbot to bombard the network with as many packets as I could in 90 minutes. During this test 529,598 packets were placed on the network for Bothunter to sort through. Again, nothing changed with Agobot. It was started on its default settings and was responsible for approximately four thousand of the traffic packets. However, for the second time we saw a drop in the number of profiles that Bothunter generated. During this test Bothunter produced 99 profiles. Figure 4.2 graphs the correlation of Bothunter profiles and the number of packets on the network.

For the seventh test we initialized Agobot, Phatbot and Sdbot on four of the five workstations. We then used a stop command to disable the scanning features of Agobot and Phatbot. With scanning disabled the three bots together generated 4,889 packets of traffic on the network. The bots were the same bots, used the same

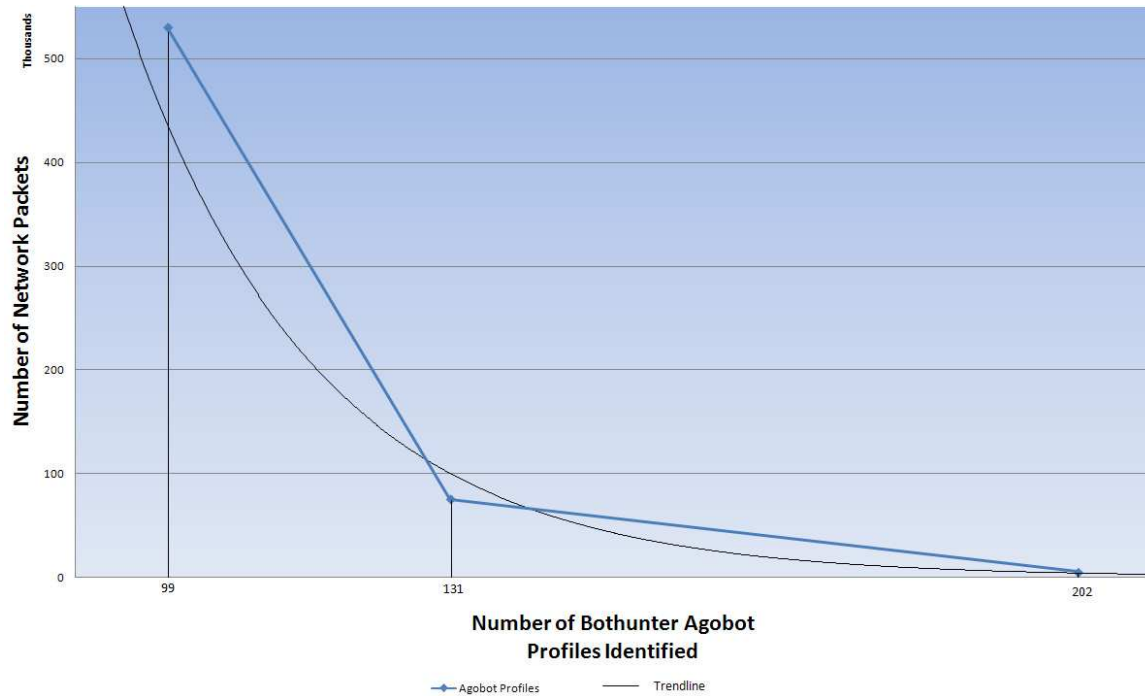


Figure 4.2: Graph showing the drop in profiles as the packet number increased

command and control channels, ports and protocols as in previous tests, but without scanning the network Bothunter failed to detect any of the bots posting 0 profiles.

One of the features of Agobot is the ability to specify how long you would like to take to scan a network. In theory, if it takes longer to scan the network, less noise and attention is warranted. Tests 8 & 9 were conducted to see if the aggressiveness of the scan affects Bothunters profile generation. The results of these two tests are interesting. Bothunter generated the exact same number of profiles for both of these two scans. Additionally, we would have expected the more aggressive scan (take less time) to produce more network packets than the more conservative scan. It could be that there is a flaw in Agobot and that the timing feature does not work the way it is documented. If Agobot does not work as advertised it could be that the two scans were almost identical. This would explain the same number of profiles generated and explain almost the same number of network packets. Another hypothesis is that the way the multiple scans alternated confused Bothunter which led to the relatively low

73 profiles when compared to Bothunters 204 profiles when only the default scanning is enabled.

Tests 10 & 11 are another attempt to see if adding network traffic affects Bothunters ability to generate bot profiles. During these two tests all of Agobots scans are enabled similar to tests 8 & 9. However this time, using packet replay software, light and heavy traffic is added. When light traffic is added a total of 74,840 packets is generated. When heavy traffic is added 468,785 packets traverse the network. However, contrary to our hypothesis that ‘bigger haystacks’ lead to finding less ‘needles’, Bothunter generated 73 profiles for both tests. Interestingly, this is the same number as in tests 8 & 9. This evidence supports the hypothesis introduced in the previous paragraph that when multiple scans are performed Bothunter is unable to put the pieces together to generate additional profiles. Another hypothesis is that Agobot does something very blatant that Bothunter picks it up no matter how many scans or how much network traffic is being push under Bothunter. It is possible with four Agobots on the network that in 90 minutes it performs this blatant action 73 times.

Finally, in one last effort to see if Bothunter would be capable of identifying Sdbot, we used a Sdbot command to perform a UDP DDoS attack on one of the workstations. Under this construct four workstations bombarded one of the workstations with UDP packets. The workstation chosen for the attack was the one that had Bothunter running on it. 93,757 packets were generated in the 90 minute test and Bothunter again failed to generate even a single profile.

Throughout all twelve tests it was also observed and recorded if the Command and Control channel was identified. Table 4.2 shows the results.

Table 4.2: Table showing C2 channel detection results

Test Description	C2 Channel Detected
Network only	N/A
Agobot only	YES
Phatbot only	NO
Rxbot only	NO
Sdbot only	NO
Agobot with light traffic	YES
Agobot with heavy traffic	NO
All bots, no scanning	NO
All bots, light scanning	NO
All bots, heavy scanning	NO
All bots, light scanning, light traffic	NO
All bots, light scanning, heavy traffic	NO

One of the claims that the authors of Bothunter makes [12] is that Bothunter is capable of tracking the C2 channel. After all twelve tests were performed Bothunter lived up to this claim only twice. Both were with Agobot, once on a quiet network and once with only light network traffic. One hypothesis is that Bothunter fails to correlate the needed pieces to identify the C2 channel when bombarded with unrelated traffic.

The tests performed lead us to conclude that Bothunter may live up to all of the claims made by its authors in only the perfect of network circumstances. Bothunter was able to detect two of the three bots. However, it should also be pointed out that on a quiet network it produced zero false positives. This gives strong credence that even one profile would be enough to alert a network administrator to the presence of a botnet on the network. Finally, it was observed that the characteristic of scanning played a large part in the detection as it failed to identify the bots when their scanning was disabled.

4.2 Testing Initialization Sequence

Once the competency of Bothunter has been determined the next step is to introduce traffic onto the network that simulates the Cybercraft initialization sequence presented in 3.4.1. Table 4.3 shows the number of profiles that Bothunter generated for each of the three scenarios.

Table 4.3: Table showing the results of the three Cybercraft initialization sequence scenarios.

Scenario Number	Scenario Description	# of Profiles Detected	# of Network Packets
1	Using HTTP traffic	0	9,182
2	Using ARP traffic	0	1,066
3	Using DNS traffic	0	1,906

The first scenario is centered around a single Cybercraft which needs to initialize and plug into a command and control structure when there are no other Cybercraft locally. In order to simulate this scenario and to have actual HTTP traffic on the research network we created five webpages (see Figure 4.3). Each webpage simulates a real world Internet webpage that could be accessed by a Cybercraft to gain information on who to contact to plug into a command and control channel, or the webpage itself could serve as the command and control channel. In order to test the stealthiness of this approach we used Wireshare to capture the packets of a workstation visiting each of the five webpages. I then put the requests in a packet replay utility and introduced other unrelated traffic. It was our hypothesis that this method would be the method most prone to detection from Bothunter. We believed this because HTTP traffic is one of the more popular methods in practice for a bot to exercise C2 on its zombies (see §2.5.1).

Our hypothesis was incorrect; Bothunter did not produce a profile during the 90 minute test. During the test the workstation that was simulating the Cybercraft

visited in turn each of the five webpages 120 times. This stands to reason that the Cybercraft could have had 600 different command and control messages and was effectively plugged into a command and control channel throughout the duration of the test.

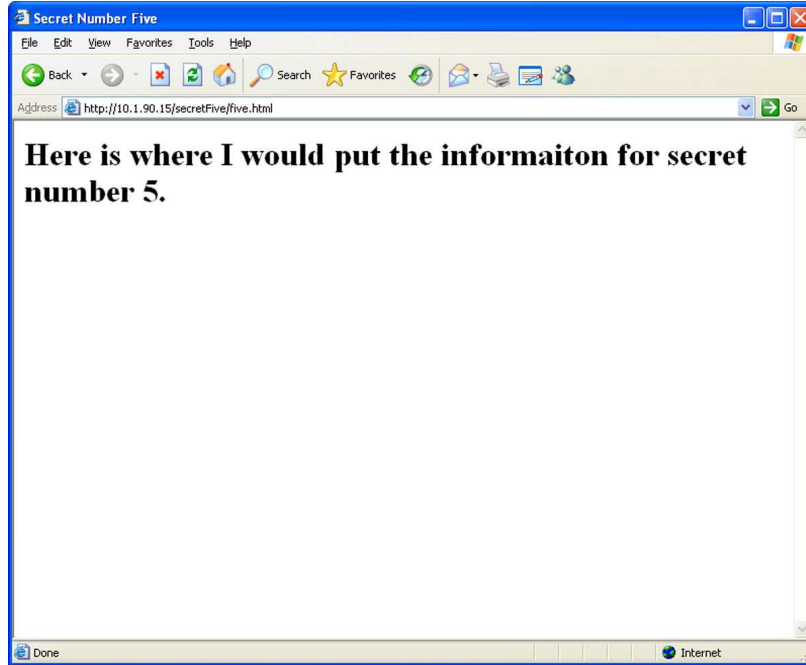


Figure 4.3: Screenshot showing webpage that could have C2 information

The second scenario assumes that a Cybercraft initializes on a network where there is already a Cybercraft C2 presence. During this scenario a Cybercraft listens and upon seeing ARP traffic with an encrypted code present contacts the sender of the ARP message. After a 48-bit encrypted handshake the workstation contacts a different Cybercraft and transfers 480 bits of information. This simulates a new node joining a Chord P2P system. Furthermore we simulated a node leaving the Chord constellation by contacting a neighboring Cybercraft (its successor) and transferring 480 bits of information. 480 bits of information is arbitrary and could be any length. This process is recorded and replayed 10 times over the 90 minute test. This test placed 1,066 packets on the network with ARP messages comprising 29.46% of those packets. Bothunter did not generate a bot profile for this scenario. During the 90

minute tests one of the Cybercraft platforms was in communication with an external source downloading 11 webpages of information. Other Cybercraft were part of a Chord system with said Cybercraft. Thus all Cybercraft in this scenario were effectively plugged into a command and control channel throughout the duration of this test.

The third and final scenario assumes the same Chord system exists as in scenario 2, but the Cybercraft that is in connection with the outside network communicates via DNS messages instead of HTTP messages. During this scenario 1,906 packets were placed on the network, 40 of those packets were DNS messages keeping the Chord constellation in constant communication to an external Cybercraft throughout the entire 90 minute test. Bothhunter did not produce a profile for any of the communications during this test. Similarly to scenario one and two, the Cybercrafts were in communication with an outside command and control channel during the entire 90 minute test.

It was the overall goal of this research to produce a methodology for a stealthy Cybercraft initialization sequence. Bothhunter, a current bot detection tool was chosen and shown that it is capable of detecting bots in real time. Three scenarios were chosen that represent three realistic Cybercraft situations. Bothhunter was presented with simulated traffic that would represent the presented methodology of a Cybercraft initialization sequence for each of the three scenarios. It was shown that Bothhunter did not generate an alert for any of the three scenarios.

4.3 Significance

This research had three goals:

- Create a methodology for a Cybercraft platform to stealthily initialize and join a command and control channel

- Test the claims made that Bothunter is capable of detecting botnets in real time.
- Test that the aforementioned initialization sequence will not be detected by the botnet detection tool Bothunter

The results that followed these three goals contribute not only to the world of academia, but help to further the Cybercraft project and the U.S. Air Force's mission of fighting and 'winning' in cyberspace. This research stands as an unbiased 3rd party validating many of the claims made by the authors of Bothunter and thier respective papers. In addition, it calls into question other claims made by the same authors. However most importantly it was the goal of this research to help further the Cybercraft project by tackling the requirement for a stealthy initialization sequence to plug into a command and control channel.

4.4 Summary

This chapter presented detailed results of the twelve experiments that were conducted against the Bothunter bot detection tool. The results of testing the presented methodology for a stealthy Cybercraft initialization sequence were presented. It was shown that Bothunter did not detect the Cybercraft initialization sequence, and that in all three scenarios the Cybercrafts enjoyed a command and control channel.

V. Conclusions and Recommendations

This section provides another challenge to the stealthiness of the Cybercraft initialization sequence discussed in Chapters 3 and 4. The concept of ARP validation is discussed with the ramifications to this research, and a solution is given to evade the ARP validation tool *XARP*. Recommendations for future work are discussed. Finally a conclusion to this thesis is presented.

5.1 *ARP Validation*

It was shown in Chapter 4 that the methodology presented in this research evades the bot detection tool Bothunter. After testing outlined in Chapter 3 was completed the methodology was examined for weaknesses. It was determined that there could be a flaw in the methodology. The methodology relies heavily on modified ARP messages. ARP messages were selected to be a fundamental part of the stealthiness of the methodology because ARP messages are frequent, they are constantly used between neighboring systems inside a network and they are frequently examined. This section brings to light how the Cybercraft initialization sequence would stand up to a tool that validates ARP requests. XArp is a tool written by Ph.D student Christoph Mayer at the Institute of Telematics, University of Karlsruhe (TH). XArp is a security application that uses advanced techniques to detect ARP-based attacks [25].

We first ran a test to baseline XArp on the same research network as the tests of Chapter 3 and 4. We ran XArp on the ‘high’ setting (see Figure 5.1) on a quiet network. XArp produced 10,344 packets in 90 minutes. During the test XArp created 35 alerts. All of the alerts were false positives that claimed that ARP reply packets came across the network without an associated request.

The next step was to run ARP the modified ARP packets across XArp. XArp flagged the packets that we were using as malicious. The reason is that we had been

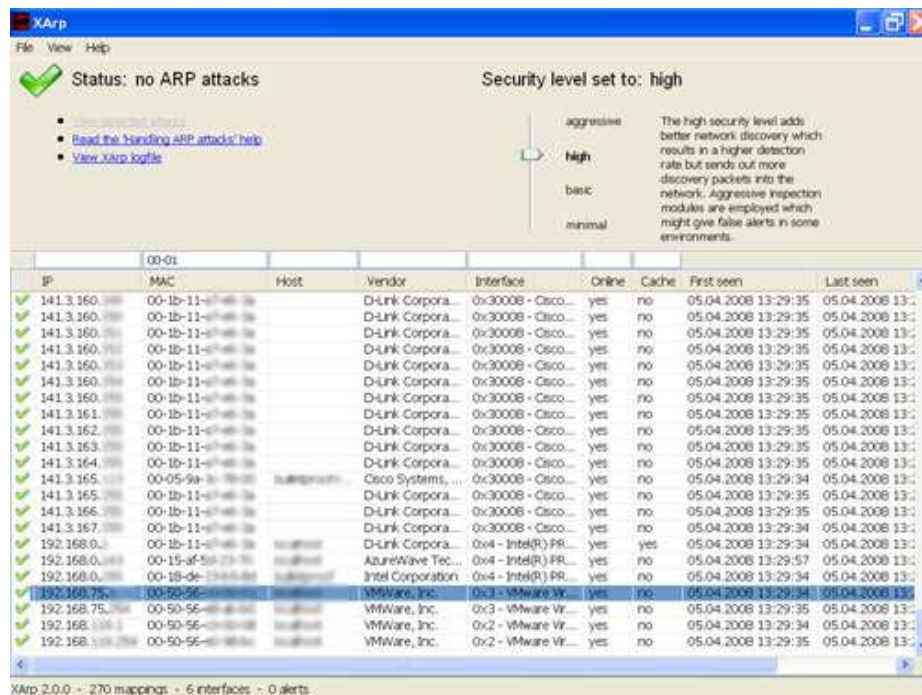


Figure 5.1: Screenshot of XArp a ARP validation utility from [25]

using the source MAC address to transfer encrypted data. XArp keeps a table of IP addresses and their associated MAC addresses. XArp was able to easily verify that our encrypted data was not the correct associated MAC address, so it flagged an Alert.

We were able to get around this limitation and successfully transmit encrypted data. Rather than use the MAC address to transmit the data the destination IP can be used. We were also able to place information in the 'Extra Data' field of the ARP request to get around XArp, however, as a general practice the 'Extra Data' field could stand out more since it is rarely used.

Thus, we were able to conclude that XArp defiantly limits what fields can be modified to send encrypted data using ARP, it is still possible to modify ARP requests to pass data between devices located on the same LAN.

5.2 *Future Work*

Bot detectors and ARP validation utilities are only two tools that are in a network administrator or forensics examiners toolbox to aid their quest of keeping malicious logic off their computers and out of their networks. There are other tools and devices that could prove to detect the presented methodology of Cybercraft initialization and reveal the command and control channel. This research showed that the given methodology could be implemented to evade the bot detection tool Bothunter. Bothunter is built on Snort, a common IDS. Future work should focus on IDS technologies and consider if there is an easy rule set that would expose the Cybercraft initialization sequence. If there is a rule or set of rules that could easily identify the Cybercraft methodology future work could add these rules to Bothunter and rerun the tests to see if it identifies this methodology.

Another area to consider is hardware-based firewalls and routers. Many of these devices can implement rules that stop packets from being forwarded. For example some Cisco devices have what Cisco calls Dynamic ARP inspection [5]. Work should be done to see if the methodology stands against these hardware tools.

Finally, it was not the purpose of this research project to build a Cybercraft platform. Future work could implement the presented pseudo code in a virtual machine based rootkit. It is expected that a proof of concept Cybercraft platform based on a VMBR would provide the same results against bot detection and ARP validation as this research, but we cannot say for certain until the proof of concept exists and is tested.

This research stands on the work of many people before me. We have incorporated many lessons learned from many different research projects into this work. There is still much to be done towards the Cybercraft project and in the field of cyberspace.

5.3 Conclusion

It was the overall goal of this research to produce a methodology for a stealthy Cybercraft initialization sequence. Bothhunter, a current bot detection tool was chosen and shown that it is capable of detecting bots in real time. Three scenarios were chosen that represent three realistic Cybercraft situations. Bothhunter was presented with simulated traffic that would represent the presented methodology of a Cybercraft initialization sequence for each of the three scenarios. It was shown that Bothhunter did not generate an alert for any of the three scenarios. The methodology for Cybercraft initialization was put against XArp, a ARP validation utility. Some variables had to be altered, but again the methodology was shown to be stealthy in the face of another real time detection utility. Finally suggestions for continuing research on this project are given.

Bibliography

1. Adams, Keith and Ole Agesen. “A comparison of software and hardware techniques for x86 virtualization”. *SIGOPS Oper. Syst. Rev.*, 40(5):2–13, 2006. ISSN 0163-5980.
2. Barford, P. and V. Yegneswaran. “An Inside Look at Botnets”. *Special Workshop on Malware Detection, Advances in Information Security*, 2006.
3. Boyd, John. “The OODA “Loop” Sketch”. Presentation, March 2006. http://www.d-n-i.net/boyd/boyds_ooda_loop.ppt [Online; accessed Jan-2009].
4. Butler, Jamie. “DKOM (Direct Kernel Object Manipulation)”. Black Hat Windows Security 2004, January 2004. <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf> [Online; accessed Jan-2009].
5. Cisco. “Configuring Dynamic ARP Inspection”, January 2009. http://www.cisco.com/en/US/docs/switches/lan/catalyst3750/software/release/12.2_40_se/configuration/guide/swdynarp.html [Online; accessed Jan-2009].
6. Cleary, Thomas and Sun Tzu. *The Art of War*. Shambhala, July 2003.
7. Cohen, F. “Computer viruses: theory and experiments”. *Comput. Secur.*, 6(1):22–35, 1987. ISSN 0167-4048.
8. Davis, Carlton R., Jose M. Fernandez, Stephen Neville, and John McHugh. “Sybil attacks as a mitigation strategy against the Storm botnet”. *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, 32–40, Oct. 2008.
9. Donley, Michael B. and Norton A. Schwartz. “Mission Statement and Priorities”. Letter, September 2008. <http://www.af.mil/library/viewpoints/jvp.asp?id=401> [Online; accessed Jan-2009].
10. Ferrie, Peter, Nate Lawson, and Thomas Ptacek. “Don’t Tell Joanna, The Virtualized Rootkit Is Dead”. Black Hat USA 2007, August 2007. https://www.blackhat.com/presentations/bh-usa-07/Ptacek_Goldsmith_and_Lawson/Presentation/bh-usa-07-ptacek_goldsmith_and_lawson.pdf [Online; accessed Jan-2009].
11. Garfinkel, Tal, Keith Adams, Andrew Warfield, and Jason Franklin. “Compatibility is not transparency: VMM detection myths and realities”. *HOTOS’07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, 1–6. USENIX Association, Berkeley, CA, USA, 2007.
12. Gu, Guofei, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. “BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correla-

- tion". *Proceedings of the 16th USENIX Security Symposium (Security'07)*. August 2007.
13. Gu, Guofei, Junjie Zhang, and Wenke Lee. "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic". *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*. February 2008.
 14. Hoglund, Greg and Jamie Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005. ISBN 0321294319.
 15. Hunt, Shannon E.M. *Developing a Reference Framework for Cybercraft Trust Evaluation*. Master's thesis, Air Force Institute of Technology, March 2008.
 16. Information Directorate, Air Force Research Laboratory. "Overview and Summary Information (AV-1) for Cybercraft Proposed Architectures", October 2008. Draft.
 17. International, SRI. "BotHunter Internet Release Software Distribution Page", December 2008. <http://www.bothunter.net> [Online; accessed Jan-2009].
 18. Kamluk, Vitaly. "The botnet business". Kaspersky Lab, May 2008. <http://www.viruslist.com/en/viruses/analysis?pubid=204792003> [Online; accessed Jan-2009].
 19. Karrels, D. *White Paper: CyberCraft C3 Architecture*. Technical report, Air Force Institute of Technology, 2008. <http://www.bunchonoobs.com/Karrels%20-%20Specialty%20Exam%20Proposal.pdf> [Online; accessed Jan-2009].
 20. King, Samuel T., Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. "SubVirt: Implementing malware with virtual machines". *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 314–327. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2574-1.
 21. Kurzban, S. "Viruses and worms - What can they do?" *SIGSAC Rev.*, 7(1):16–32, 1989. ISSN 0277-920X.
 22. Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. ISSN 0164-0925.
 23. Mandia, Kevin. *Incident Response: Investigating Computer Crime*. McGraw-Hill Professional, 2001. ISBN 0072131829.
 24. Mann, Andi. *Virtualization 101: Technologies, Benefits, and Challenges*. White paper, Enterprise Management Associates, August 2006. http://www.emausa.com/web/EMA_Virtualization_WP-0806.pdf.
 25. Mayer, Christoph. "chrismc", December 2008. <http://www.chrismc.de/development/xarp/index.html> [Online; accessed Jan-2009].
 26. McClure, Stuart, Joel Scambray, and George Kurtz. *Hacking Exposed 5th Edition (Hacking Exposed)*. McGraw-Hill Osborne Media, 2005. ISBN 0072260815.

27. McDonald, J. Todd. *Lecture Notes in Computer Science*, chapter Hybrid Approach for Secure Mobile Agent Computations. Springer Berlin, Heidelberg, 2006.
28. McDonald, T., Peterson B., D. Karrels, Andel T., and Raines R. "Guarding the Cybercastle in 2020". *IAnewsletter*, 11(3), 2008.
29. Medley, Douglas P. *Virtualization Technology Applied to Rootkit Defense*. Master's thesis, Air Force Institute of Defense, March 2007.
30. Moore, Andrew P., Dawn M. Cappelli, and Randall F. Trzeciak. *The Big Picture of Insider IT Sabotage Across U.S. Critical Infrastructures*. Technical report, Software Engineering Institute, Carnegie Mellon, May 2008. http://www.cert.org/insider_threat/ [Online; accessed Jan-2009].
31. Peterson, Gilbert L. and Daniel R. Karrels. "Survey of Structured Peer-to-Peer Overlay Networks", July 2008. Air Force Institution of Technology.
32. Phister, Paul W., Dan Fayette, and Emily Krzysiak. "CyberCraft: Concept Linking NCW Principles with the Cyber Domain in an Urban Operational Environment". *Proceedings of ICCRTS*. International Command and Control Research and Technology Symposium, June 2005.
33. Reagan, President Ronald. "Farewell Address to the Nation", January 1989. <http://www.reaganfoundation.org/reagan/speeches/farewell.asp> [Online; accessed Jan-2009].
34. Rutkowska, Joanna. "System Virginity Verifier - Defining the Roadmap for Malware Detection on Windows System". Hack In The Box Security Conference Presentation, September 2005. http://invisiblethings.org/papers/hitb05_virginity_verifier.ppt [Online; accessed Jan-2009].
35. Rutkowska, Joanna. "Introducing Stealth Malware Taxonomy". Black Hat Federal Conference Presentation, November 2006. <http://invisiblethings.org/papers/malware-taxonomy.pdf> [Online; accessed Jan-2009].
36. Rutkowska, Joanna. "Subverting Vista Kernel For Fun And Profit". Black Hat USA 2006, August 2006. <http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf> [Online; accessed Jan-2009].
37. Rutkowska, Joanna. "Virtualization Detection vs. Blue Pill Detection". Internet Blog, August 2007. <http://theinvisiblethings.blogspot.com/2007/08/virtualization-detection-vs-blue-pill.html> [Online; accessed Jan-2009].
38. Rutkowska, Joanna and Alexander Tereshkin. "Bluepillling the Xen Hypervisor". Black Hat USA 2008, August 2008. <http://invisiblethingslab.com/resources/bh08/part3.pdf> [Online; accessed Jan-2009].
39. Skoudis, Edward and Tom Liston. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. ISBN 0131481045.

40. Stevens, Michael. *Use of Trust Vectors in Support of the CyberCraft Initiative*. Master's thesis, Air Force Institute of Technology, March 2007.
41. Tanenbaum, Andrew S. and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 0132392275.
42. Terro. "Outdoor Liquid Ant Bait". Internet Graphic, 2009. http://www.terro.com/products.php?product=outdoor_ant_bait [Online; accessed Jan-2009].
43. United States. *Air Force Basic Doctrine: Air Force Doctrine Document 1*. Headquarters Air Force Doctrine Center, Maxwell AFB, AL, 2003.
44. Xen. *Xen: Enterprise Grade Open Source Virtualization; Inside Xen 3.2 A Xen White Paper*. White paper, Xen.org, June 2006. <http://xen.org/files/xenWhitePaper3.2.pdf>.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 26-03-2009		2. REPORT TYPE Master's Thesis			3. DATES COVERED (From — To) Sept 2007 — Mar 2009	
4. TITLE AND SUBTITLE Using Covert Means To Establish Cybercraft Command And Control				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Bradley D. Sevy, Capt, USAF				5d. PROJECT NUMBER 08-198		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/09-07	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Robert L. Herklutz Program Manager: Security and Information Operations AFUSR Suite 325, Room 3112 875 N. Randolph St. Arlington, VA 22203-1768 E-mail: robert.herklutz@afosr.af.mil 703-696-6565 Fax 703-696-8450					10. SPONSOR/MONITOR'S ACRONYM(S) AFOSR/NL	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approval for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT The Air Force Research Laboratory (AFRL) Information Directorate (RI) is researching a next generation network defense architecture, called Cybercraft, that will provide automated and trusted cyber defense capabilities for AF network assets. In this research, we consider the issues of how to protect or obfuscate command and control aspects of the system. In particular, we present a methodology to hide aspects of Cybercraft platform initialization in context to formation of hierarchical, peer-to-peer groups that collectively form the Cybercraft network. This research will subject Bothunter to a series of tests to validate these claims. We use a leading bot detection utility, Bothunter, and an ARP validation tool, XArp, to build a case for the effectiveness of our approach. We present three scenarios that correlate to how we believe Cybercraft platforms will be integrated in the future and consider stealthiness in terms of these representative tools.						
15. SUBJECT TERMS cybercraft, malware, botnet, botnet detection						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 84	19a. NAME OF RESPONSIBLE PERSON Lt Col J. Todd McDonald	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (include area code) (937) 255-3636, ext 4639; e-mail: Jeffrey.McDonald@afit.edu	